
Донецкий национальный технический университет
А. Я. Аноприенко, С. В. Иваница
Интернет-технологии для студентов и преподавателей
Развитие технологий гипертекста
Язык гипертекстовой разметки
Распространенные элементы HTML
Каскадные теблицы стилей CSS
1921
ТЕОРИЯ
ПРАКТИКА
ДОНЕЦКИЙ
НАЦИОНАЛЬНЫЙ
ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
ГОУВПО
ДонНТУ
УНИТЕХ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

А. Я. Аноприенко ♦ С. В. Иваница

ИНТЕРНЕТ-ТЕХНОЛОГИИ ДЛЯ СТУДЕНТОВ И ПРЕПОДАВАТЕЛЕЙ

Учебное пособие
для обучающихся образовательных учреждений
высшего профессионального образования

КНИГА ВТОРАЯ

- Развитие технологий гипертекста
- Язык гипертекстовой разметки
- Распространенные элементы HTML
- Каскадные таблицы стилей CSS

Издание приурочено к 100-летию Донецкого национального технического университета

Донецк ♦ 2021

УДК 004.738.5(07)
ББК 32.973.202-04я73
А69

Рекомендовано Ученым советом ГОУВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» в качестве учебного пособия для обучающихся образовательных учреждений высшего профессионального образования (протокол № 1 от 26 февраля 2021 г.)

Рецензенты:
Аверин Геннадий Викторович — доктор технических наук, профессор, заведующий кафедрой компьютерных технологий ГОУВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ»;
Метлов Константин Леонидович — доктор физико-математических наук, старший научный сотрудник ГУ «Донецкий физико-технический институт им. А. А. Галкина»;
Зори Сергей Анатольевич — доктор технических наук, заведующий кафедрой программной инженерии ГОУВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ».

Авторы:
Аноприенко Александр Яковлевич — кандидат технических наук, профессор кафедры компьютерной инженерии, ректор ГОУВПО «ДОННТУ»;

Иваница Сергей Васильевич — кандидат технических наук, доцент кафедры компьютерной инженерии, директор Центра информационных компьютерных технологий ГОУВПО «ДОННТУ».

Аноприенко, А. Я.

А69 Интернет-технологии для студентов и преподавателей : учеб. пособие для обучающихся образоват. учреждений высш. проф. образования : книга вторая / А. Я. Аноприенко, С. В. Иваница ; ГОУВПО «ДОННТУ». — Донецк : УНИТЕХ, 2021. — 268 с. : ил.

ISBN 978-966-8248-95-5

Учебное пособие написано на основе материалов курса «Интернет-технологии» для магистрантов, читаемого в Донецком национальном техническом университете с 2000 года. Эта книга — вторая из серии «Интернет-технологии для студентов и преподавателей», в которую вошел материал о языке гипертекстовой разметки HTML и каскадных таблицах стилей CSS.

Издание предназначено для старших школьников и студентов, магистрантов и аспирантов, молодых ученых, преподавателей, а также для всех интересующихся интернет-технологиями.

ISBN 978-966-8248-95-5

УДК 004.738.5(07)
ББК 32.973.202-04я73

СОДЕРЖАНИЕ

Введение.....7

ГЛАВА 1. ОСНОВЫ ЯЗЫКА РАЗМЕТКИ ГИПЕРТЕКСТА HTML 11

1.1. Развитие технологий гипертекста.....11

1.1.1. Основные понятия и определения. Классификация гиперссылок.....12

1.1.2. Три поколения исторического развития гипертекста..17

1.1.3. Развитие языка гипертекстовой разметки HTML 30

1.2. Язык гипертекстовой разметки 38

1.2.1. Принципы теговой модели.....42

1.2.2. Технология «Клиент-Сервер» 46

1.2.3. Обработка веб-документов в браузере.....48

1.2.4. Структура документа HTML.....51

1.2.5. Обязательные элементы HTML 58

1.3. Основы HTML 63

1.3.1. Дерево HTML-документа 64

1.3.2. Таблицы элементов и атрибутов....69
1.3.3. Адресация в HTML....72
1.3.4. Организация гиперссылок....77
1.3.5. Универсальные атрибуты....81
1.3.6. Комментарии в HTML....83
1.4. Элементы HTML....85
1.4.1. Основные элементы для оформления текстов....85
1.4.2. Дополнительные элементы для оформления текстов..93
1.4.3. Графические элементы....96
1.4.4. Блочные и строчные элементы HTML....99
1.4.5. Универсальные элементы div и span....102
1.4.6. Организация списков в HTML....104
1.4.7. Таблицы в HTML....107
Контрольные вопросы для самопроверки знаний....114
Список литературы к главе 1....116

ГЛАВА 2. КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ CSS.....119

2.1. Общие сведения о CSS....119
2.1.1. Краткая история развития CSS....121
2.1.2. CSS Zen Garden....124
2.2. Основы синтаксиса, методы определения, каскадирование стилей....125
2.2.1. Основы синтаксиса CSS....129
2.2.2. Формы записи стилей CSS....131
2.2.3. Методы определения таблицы стилей....132
2.3. Виды селекторов CSS....138
2.3.1. Селекторы типов....138
2.3.2. Селекторы классов....140
2.3.3. ID-селекторы....144
2.3.4. Стилизация групп тегов....145
2.3.5. Контекстные селекторы....147

2.3.6. Дочерние селекторы 148
2.3.7. Соседние селекторы 149
2.3.8. Селекторы атрибутов 151
2.3.9. Псевдоклассы 155
2.4. Единицы измерения, способы задания цветов, шрифтовое оформление 159
2.4.1. Единицы измерения в CSS 159
2.4.2. Способы задания цветов в CSS 162
2.4.3. Шрифтовое оформление в CSS 168
2.4.4. Оформление текстов в CSS 183
2.5. Блочная модель CSS: позиционирование, выравнивание и обтекание 191
2.5.1. Управление фоном содержимого блока 194
2.5.2. Позиционирование 210
2.5.3. Выравнивание и обтекание 216
2.6. Преобразования, переходы и анимация в CSS 222
2.6.1. Преобразования 223
2.6.2. Переходы 231
2.6.3. Анимация 239
Контрольные вопросы для самопроверки знаний 249
Список литературы к главе 2 252

Заключение 255

Портал магистров ДонНТУ (masters.donntu.org) 257
Избранные коллективные фото магистров факультета КНТ 258

Учебно-методическое обеспечение курса «Интернет-технологии» 262

ВВЕДЕНИЕ

В наше время информационные технологии все глубже проникают в жизнь каждого человека, закрепляя за собой статус неотъемлемого компонента в развитии цивилизованного общества в целом. Важную роль играет стремительный рост и увеличение технологических масштабов интернет-технологий, которые выражены не только как источники информации, но являются универсальной средой для общения, обучения, проведения досуга. Количество пользователей Интернетом растет с каждым днем. С еще большей скоростью появляются новые сайты и веб-порталы (или обновляются существующие), покрывающие информационно-технологические потребности всех без исключения сфер человеческой деятельности.

Поэтому роль Интернета в современном обществе трудно переоценить, учитывая его характеристики воздействия на общество, которые включают в себя:

1) средства мгновенного поиска: в настоящее время полезной и познавательной информацией Всемирной паутины пользуются все, от школьников до президентов;
2) коммуникационную среду: через Интернет можно связаться с человеком почти в любой точке мира, отправлять и получать письма за секунды, мгновенно делать различные покупки;

3) коммерческую площадку: Интернет в современной жизни человека это не только развлечения, получение информации и общение, но и реальное рабочее место, — уже никого не удивляют такие понятия как интернет-предприниматель, онлайн-бизнес, сетевой маркетинг, фрилансинг.

При лавинообразном росте информационных ресурсов, принципиально важной является также возрастающая автоматизация оценки эффективности с помощью различных рейтинговых систем, что в конечном итоге повышает объективность, эффективность и значимость оцениваемых ресурсов.

Поэтому с каждым годом все большую актуальность набирает **учебная дисциплина «Интернет-технологии»**, изучаемая магистрантами всех направлений подготовки и читаемая авторами этого учебного пособия — преподавателями кафедры компьютерной инженерии факультета компьютерных наук и технологий ГОУВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДОННТУ). По состоянию на 2021 год, дисциплина «Интернет-технологии» входит в перечень образовательных дисциплин для магистрантов вот уже на протяжении 20 лет!

Содержание дисциплины раскрывается в следующих темах:

1. Использование Интернета как качественно нового источника информации.
2. Разработка тематических электронных библиотек и списков ссылок на документы или информационные ресурсы.
3. Разработка гипертекстовых документов.
4. Работа с графической информацией в Интернете.
5. Программирование и отладка в среде браузера.
6. Методы и технологии оперативного профессионального общения в Интернете.
7. Применение веб-типографики для оформления текстов в Интернете.
8. Изучение и эффективное использование современного программного обеспечения для просмотра и верстки сайтов.

В данном учебном пособии рассматривается часть вышеперечисленных тем, однако *вся серия учебных изданий к дисциплине «Интернет-технологии для студентов и преподавателей» полностью охватит все содержание дисциплины «Интернет-технологии»*.

Серия взаимосвязанных лабораторных работ направлена на создание к концу изучаемого курса персонального тематического сайта, основой содержания которого являются систематизированные материалы по теме выпускной работы магистра, подобранные и обработанные с использованием интернет-технологий в процессе выполнения лабораторных работ по курсу. Разработанные в рамках данного курса сайты размещаются на **портале магистров ДонНТУ** (**masters.donntu.org**), который является одним из самых посещаемых научно-образовательных сайтов образовательных организаций в Донецкой Народной Республике.

Эта книга — вторая из книг серии «Интернет-технологии для студентов и преподавателей», в которую вошел материал о развитии гипертекстовых технологий и появившегося на их основе языка гипертекстовой разметки. Также читателям предложен обширный материал о каскадных таблицах стилей, при изучении которых детально рассматриваются основные принципы и подходы к созданию динамических элементов веб-страницы.

Книга содержит две тематические главы:

Глава 1. Основы языка разметки гипертекста HTML.

В главе рассматривается развитие технологий гипертекста, три поколения исторического развития гипертекста, развитие языка гипертекстовой разметки HTML. Изучение языка HTML включает рассмотрение принципов теговой модели, технологии «Клиент-Сервер», детальное изучение процесса обработки веб-документов в браузере. В подглаве, излагающей структуру документа HTML, говорится об обязательных элементах HTML, дерева элементов, работе с таблицами элементов и атрибутов, адресации HTML для организации гиперссылок и универсальных атрибутах. При изучении основ HTML, читателю предлага-

ется материал, позволяющий изучить основные и дополнительные элементы HTML для оформления текстов, графические элементы, элементы, участвующие в организации списков и таблиц на веб-страницах.

🖫 **Глава 2. Каскадные таблицы стилей CSS.**

В главе рассматривается краткая история развития, основы синтаксиса, методы определения и каскадирование стилей CSS. Детально изложены методы определения таблицы стилей, виды селекторов, псевдоклассы. Описаны принятые в CSS основные единицы измерения, способы задания цветов и шрифтовое оформление. Особое внимание уделено блочной модели CSS, подходам к применению позиционирования, выравнивания и обтекания. Динамическая составляющая CSS в материалах главы предлагает к изучению основные стилевые свойства для реализации преобразований, переходов и анимации в CSS.

Каждая глава сопровождена исчерпывающими **примерами** с фрагментами разметки HTML или таблиц стилей CSS для подкрепления теоретического материала практическими реализациями. В конце глав размещены **список контрольных вопросов** и **перечень используемой литературы**, в который также включены все ссылки на рассматриваемые в разделе интернет-сервисы, порталы и сайты.

В приложении в конце книги показана главная страница **портала магистров** ДонНТУ, **коллективные фото магистров** факультета компьютерных наук и технологий (КНТ) различных годов выпуска, а также описание изданий, составляющих **учебно-методическое обеспечение** курса «Интернет-технологии».

Авторы данного пособия подготовили и изложили материал таким образом, чтобы он был в равной степени понятен и полезен как магистрантам и аспирантам, так и старшим школьникам и студентам. Авторы уверены, что эта книга будет также интересна и полезна как молодым ученым, так и преподавателям всех квалификационных уровней.

1 ОСНОВЫ ЯЗЫКА РАЗМЕТКИ ГИПЕРТЕКСТА HTML

1.1. Развитие технологий гипертекста

Важнейшие свойства и особенно реализации гипертекста прочно вошли в повседневную жизнь в виду глобализации веб-технологий. Однако гипертекст не всегда был выражен теми технологическими реализациями, которые сегодня кажутся естественными и вполне очевидными. В данном разделе рассматривается эволюция гипертекста в историко-технологическом контексте. Начнем с простейших определений:

> **Гипертекст** (англ. *hypertext*) — специальным образом организованный текст, позволяющий пользователю осуществлять переход к связанным ресурсам по указателям (ссылкам).
>
> **Гипертекстовая система** — информационная система, способная хранить информацию в виде электронного текста, позволяющая устанавливать электронные связи между любыми «информационными единицами», хранящимися в ее памяти.

Иными словами, гипертекст представляет такую организацию текстовой информации, внутри которой установлены смысловые связи между ее различными фрагментами. Такие связи называют

гиперсвязями, а место в тексте, обеспечивающее гиперсвязь, — *гипертекстовой ссылкой (гиперссылкой)*:

> **Гиперсвязь** — организация динамической связи между двумя объектами ресурса сети Интернет посредством гиперссылки.
>
> **Гиперссылка** (англ. *hyperlink*) — часть гипертекстового документа, ссылающаяся на другой элемент (команда, текст, заголовок, примечание, изображение) в самом документе, на другой объект (файл, каталог, приложение), расположенный на локальном диске или в компьютерной сети, либо на элементы этого объекта.

1.1.1. Основные понятия и определения. Классификация гиперссылок

В настоящее время в компьютерной сфере термин «гипертекст» относят в равной мере к разным объектам:

1) так называют особый метод построения информационных систем, обеспечивающих прямой доступ к данным с сохранением естественных (или специальным образом предустановленных) логических связей между ними;
2) это определенная система представления гипермедийной информации (текстовой и мультимедийной) в виде сети связанных между собой информационных единиц;
3) это особый универсальный компьютерный программный интерфейс, отличительными чертами которого являются его интерактивность и чрезвычайно высокая степень адаптивности по отношению к психологии пользователя.

Гипертекстовая технология позволяет определять, выбирать вариант актуализации информации гипертекста в зависимости от информационных потребностей пользователя и его возможностей, уровня подготовки. При работе с гипертекстовой системой, пользователь имеет возможность просматривать фрагменты текста в необходимом ему порядке, а не последовательно,

как это принято при постраничном чтении. Достигается это путем создания гипертекстовых ссылок.

Гиперссылка в рамках Интернет-технологий представляет собой фрагмент документа HTML и его базовый элемент:

- указывающий на другой файл, который может быть расположен в Интернете;
- содержащая полный путь (URL-адрес) к этому файлу.

В настоящее время наибольшее распространение гипертекст как принцип интерактивной обучающей среды получил при создании электронных обучающих средств. Представление учебного материала в гипертекстовой форме существенно изменяет структуру и расширяет возможности электронного текста.

С развитием компьютерных средств мультимедиа гипертекст начал превращаться в более наглядную информационную форму, получившую название **гипермедиа**, *т. е. структура, содержащая текст, аудио- и видеофрагменты, соединенные ссылками в соответствии с логикой сюжета*. Технология гипермедиа позволяет с помощью программного обеспечения и технологических средств объединить гипертекст, графические (статические) изображения, анимационные фрагменты, аудио- и видеозаписи.

Таким образом, в качестве гиперссылки для пользователя сети Интернет может выступать графическое изображение, видео или текст на сайте, в письме электронной почты или в какомлибо электронном документе, устанавливающие связь и позволяющие переходить к другим объектам Интернета.

Информация, идущая от разных объектов, должна быть согласованной, чтобы ее воздействие не ослабляло, а усиливало восприятие. Текст, сопровождаемый рисунком, воспринимается лучше, образ, составленный путем наложения фонового изображения, динамического объекта, анимации, звукового и речевого сопровождения способен проявить синергический взаимоусиливающий эффект при его восприятии.

Итак, **основная идея гипертекстовых систем** заключается в концепции автоматически поддерживаемых связей между раз-

личными фрагментами информации (информационными единицами). Поддержка таких связей позволяет организовывать *«нелинейные» информационные структуры*. В качестве гиперссылок в электронном тексте могут выступать:

- ссылки на словарь терминов и понятий (выделение ключевых слов в тексте);
- ссылки на персоналии (портреты и краткие биографические сведения);
- ссылки на статические иллюстрации (изобразительные и условно-графические, в т. ч. схемы, таблицы и т. д.)
- ссылки на мультимедийные элементы (анимации, аудио- и видеофрагменты);
- ссылки на хрестоматийный или дополнительный материалы;
- ссылки на структурные элементы текста (оглавление, номер темы, пункт и подпункт, список вопросов для закрепления и устных развернутых ответов и др.);
- ссылки на список монографий, учебной и научной литературы (приводится в конце темы или всего курса);
- ссылки на список организаций;
- ссылки на список исторических событий или дат (хронологический указатель);
- ссылки на список географических названий;
- ссылки на Интернет-ресурсы (образовательные сайты, электронные библиотеки, мультимедийные приложения и др.).

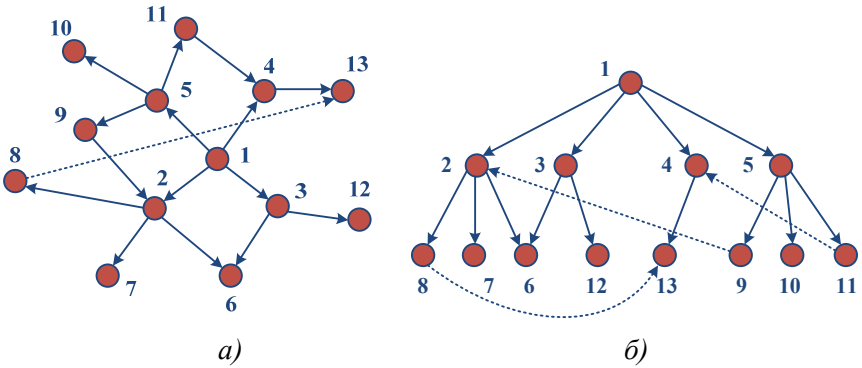
При создании гипертекста, гиперссылки, как правило, выделяются из общей массы текста. Для этого можно использовать следующие приемы:

- изменение цвета гиперссылки относительно общего цвета текста;
- изменение начертания шрифта для гиперссылки;
- увеличение или уменьшение размера шрифта;
- выделение подчеркиванием, курсивом или жирным шрифтом;

– применение различных комбинаций вышеперечисленных способов.

В разнообразных текстовых редакторах и программах при создании веб-страниц гиперссылки выделяются различными способами. Например, в программных продуктах MS Office ссылки автоматически выделяются синим цветом и подчеркиваются.

Формализация структуры гипертекста обычно осуществляется при помощи отображения ее в виде ориентированного графа, в котором точками обозначаются гипертекстовые (информационные) узлы (англ. *hypernode*), а стрелками — связи между ними (гиперссылки). Пример такого изображения в сравнении с более традиционной («древовидной») схемой отображения информации представлен на рис. 1.1 [1].



а) *б)*

Рис. 1.1. Схематичное представление гипертекстового пространства в виде ориентированного графа (а) и дерева зависимостей (б)

Информационные гипертекстовые узлы обычно представляют собой некое концептуальное утверждение или выражают одну определенную идею. Эта информация может быть поразному оформлена и структурирована в зависимости от своей функциональной семантики (например, утверждение, разъяснение, справочная информация, развивающая и детализирующая то или иное утверждение, иллюстративная информация, экспериментальные результаты, наблюдения, фактическая информа-

ция и выводы, информация прикладного характера и ее семантико-синтаксическое описание).

Информационные узлы образуют **единое гипертекстовое пространство** благодаря связям между ними, которые устанавливаются создателями гипертекста априорно в соответствии с некоторыми собственными концептуальными представлениями через систему гиперссылок. Тот узел, который является исходным для некоторой ссылки, называется **узлом-адресантом**, или **референциальным информационным узлом**. Узел, в который ведет та или иная ссылка, называется **узлом-адресатом ссылки**, или **референтом (сигнификативным) узлом**. Любой фрагмент информационного узла может быть связан с другим узлом, его частью или со своим любым другим фрагментом с помощью так называемых **якорей** (англ. *anchor*), или специальных не видимых читателю **маркеров**, которые помещаются в определенных частях текста (рисунка, схемы, аудио-, видеофайла и т. п.) [2].

Содержимое информационного узла или его фрагмента выводится на экран компьютера с *помощью активизации гиперссылки*. Они могут быть:

- **однонаправленные** — ведущие из одного гипертекстового узла в другой;
- **двунаправленные** — обеспечивающие перемещение между двумя связанными информационными фрагментами в обе стороны.

До повсеместного распространения браузеров это различие между типами гиперссылок очень активно использовалось в обучающих компьютерных системах. В настоящий момент *все гиперссылки в сети Интернет являются двунаправленными*, так как возможность возврата на любой из нескольких предыдущих этапов просмотра веб-страницы заложена в самом браузере.

Гиперссылки могут быть также классифицированы *в зависимости от их структурно-функциональной роли в общей системе гипертекстового пространства*:

- **референциальные ссылки** для установления отношений перекрестного цитирования;

– **иерархические ссылки** для установления отношений типа «родитель — ребенок».

Эта классификация в настоящее время активно используется для наиболее эффективной системы организации информационного поиска в Интернете во всех наиболее известных поисковых системах.

1.1.2. Три поколения исторического развития гипертекста

Следует отметить, что теоретические предпосылки для создания гипертекста были заложены еще за полвека до создания первого сайта. В хронологическом контексте историю гипертекста можно условно разделить на три поколения (рис. 1.2).

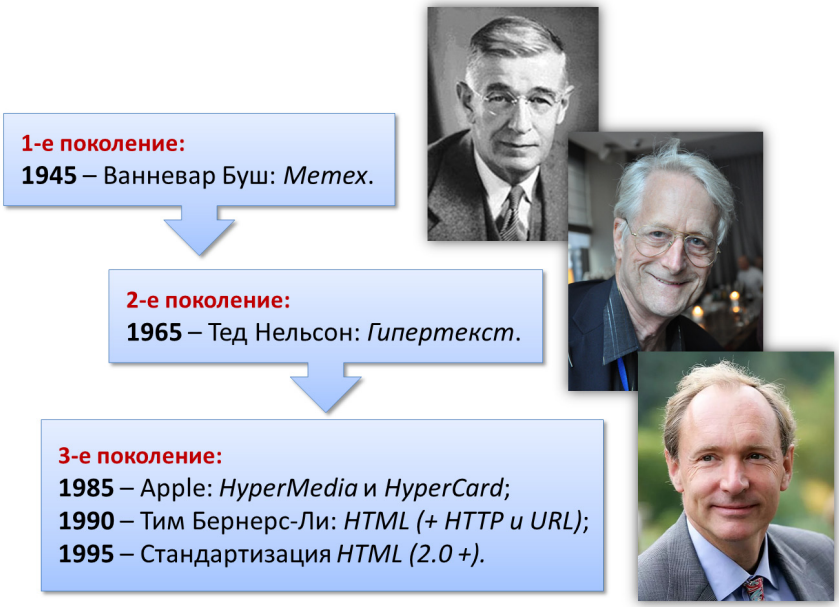


Рис. 1.2. Три поколения развития гипертекста и ключевые ученые-разработчики (вверху (1-е поколение) – Ванневар Буш; по середине (2-е поколение) – Тед Нельсон; внизу (3-е поколение) – Тим Бернерс Ли)

Первое поколение развития началось с формирования идеи гипертекста, которая была выдвинута *Ванневаром Бушем* (англ. *Vannevar Bush*) в 1945 году в предложениях по созданию **электромеханической информационной системы MEMEX**.

В 1940 году американский инженер и разработчик аналоговых компьютеров Ванневар Буш был назначен председателем Государственного комитета оборонных исследований США и советником президента по науке при президенте Рузвельте. В 1944 году Рузвельт запрашивает у Буша рекомендации о том, какие уроки Второй мировой войны следует извлечь США, на что получает ответ: *«…государственные интересы в области науки и образования могут быть наилучшим образом достигнуты созданием Национального фонда науки»*.

В период 1941–1947 гг. Ванневар Буш возглавлял бюро научных исследований и разработок при правительстве США и являлся председателем Комитета по военной политике, занимаясь координацией усилий научного сообщества (6000 ведущих ученых страны) в целях военной обороны, разработкой ядерного оружия и *Манхэттенским проектом*[1]. В 1950 г. В. Буш возглавил созданный к тому времени *Национальный фонд науки* (NSF).

В 1945 году Ванневар Буш в нашумевшей статье «Как мы можем мыслить» (англ. *As We May Think*) (рис. 1.3) предлагает прообраз гипертекстового устройства Memex: *«Человек построил столь сложную цивилизацию, что он нуждается в механизмах обработки данных, которые уже не вмещаются в его ограниченную память…»* [3].

Устройство *MEMEX* (англ. *MEMory EXtender — «расширитель памяти»*) в представлении Буша призвано помочь человеку хранить все его книги, все его записи и все его коммуникации

[1] «Проект Манхэттен» (англ. *Manhattan Project*) — кодовое название программы США по разработке ядерного оружия (начата в 1942 году), в рамках которой были созданы три атомные бомбы: плутониевая «Штучка» (Gadget) (взорвана при первом ядерном испытании), урановый «Малыш» (Little Boy) (сброшена на Хиросиму 6 августа 1945 года) и плутониевый «Толстяк» (Fat Man) (сброшена на Нагасаки 9 августа 1945 года).

с другими людьми (рис. 1.4). Устройство выглядит как обычный стол, на котором клавиатура, кнопки и рычажки. Небольшая часть стола занята данными в виде микрофильмов, остальная часть — рабочий механизм.

A SCIENTIST OF THE FUTURE RECORDS EXPERIMENTS WITH A TINY CAMERA FITTED WITH UNIVERSAL-FOCUS LENS. THE SMALL SQUARE IN THE EYEGLASS AT THE LEFT SIGHTS THE OBJECT

AS WE MAY THINK

A TOP U. S. SCIENTIST FORESEES A POSSIBLE FUTURE WORLD IN WHICH MAN-MADE MACHINES WILL START TO THINK

by VANNEVAR BUSH

DIRECTOR OF THE OFFICE OF SCIENTIFIC RESEARCH AND DEVELOPMENT

Condensed from the *Atlantic Monthly*, July 1945

This has not been a scientists' war; it has been a war in which all have had a part. The scientists, burying their old professional competition in the demand of a common cause, have shared greatly and learned much. It has been exhilarating to work in effective partnership. What are the scientists to do next?

For the biologists, and particularly for the medical scientists, there can be little indecision, for their war work has hardly required them to leave the old paths. Many indeed have been able to carry on their war research in their familiar peacetime laboratories. Their objectives remain much the same.

It is the physicists who have been thrown most violently off stride, who have left academic pursuits for the making of strange destructive gadgets, who have had to devise new methods for their unanticipated assignments. They have done their part on the devices that made it possible to turn back the enemy. They have worked in combined effort with the physicists of our allies. They have felt within themselves the stir of achievement. They have been part of a great team. Now one asks where they will find objectives worthy of their best.

* * *

There is a growing mountain of research. But there is increased evidence that we are being bogged down today as specialization extends. The investigator is staggered by the findings and conclusions of thousands of other workers—conclusions which he cannot find time to grasp, much less to remember, as they appear. Yet specialization becomes increasingly necessary for progress, and the effort to bridge between disciplines is correspondingly superficial.

Professionally our methods of transmitting and reviewing the results of research are generations old and by now are totally inadequate for their purpose. If the aggregate time spent in writing scholarly works and in reading them could be evaluated, the ratio between these amounts of time might well be startling. Those who conscientiously attempt to keep abreast of current thought, even in restricted fields, by close and continuous reading might well shy away from an examination calculated to show how much of the previous month's efforts could be produced on call.

Mendel's concept of the laws of genetics was lost to the world for a generation because his publication did not reach the few who were capable of grasping and extending it. This sort of catastrophe is undoubtedly being repeated all about us as truly significant attainments become lost in the mass of the inconsequential.

Publication has been extended far beyond our present ability to make real use of the record. The summation of human experience is being expanded at a prodigious rate, and the means we use for threading through the consequent maze to the momentarily important item is the same as was used in the days of square-rigged ships.

But there are signs of a change as new and powerful instrumentalities come into use. Photocells capable of seeing things in a physical sense, advanced photography which can record what is seen or even what is not, thermionic tubes capable of controlling potent forces under the guidance of

Рис. 1.3. Страница статьи В. Буша «Как мы можем мыслить» [4]

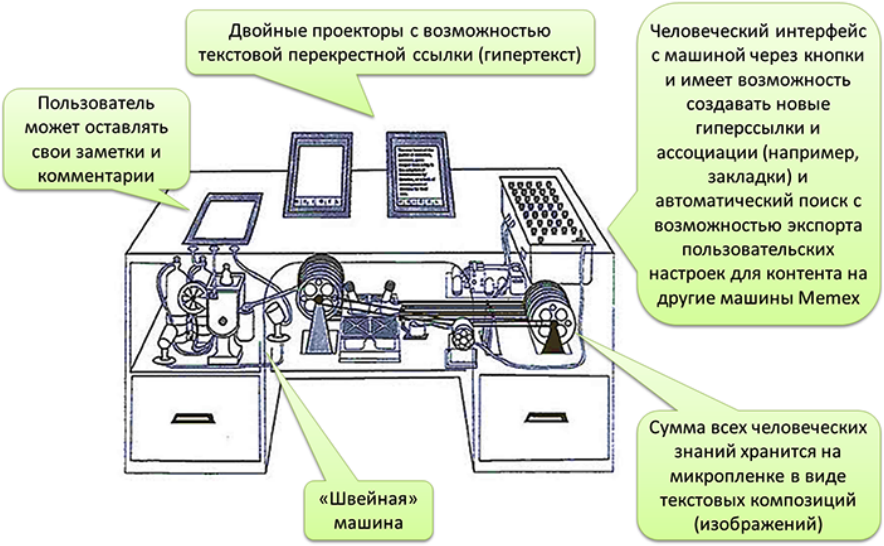


Рис. 1.4. Концепция MEMEX в представлении Буша

Книги всех типов, картинки, газеты могут быть немедленно получены и включены в систему. Эта система использует индексирование — если человек хочет получить доступ к книге — он набирает необходимый код на клавиатуре и нужная книга или страница возникает перед ним на экране MEMEX.

Когда пользователь строит ассоциативную цепочку между двумя документами, то он записывает название цепочки в книгу кодов (рис. 1.5). Сохраненные цепочки могут быть доступны пользователю в любое время. Они образуют совершенно новую книгу, которая хранится внутри MEMEX и может быть вызвана из его памяти и через много лет.

Несмотря на то, что Буш был советником по науке президента Рузвельта, его призывы к физической реализации электромеханической информационной системы оказались безуспешными — идея создания MEMEX так и не была реализована.

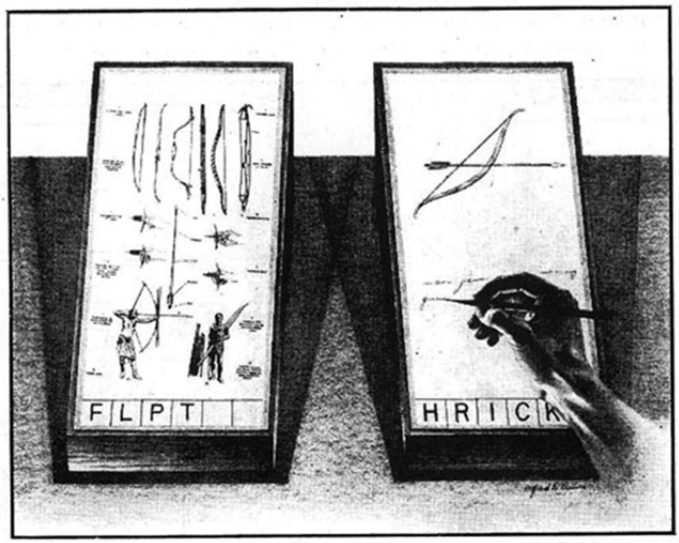


Рис. 1.5. Предполагаемая работа с сохраненными цепочками в книгу кодов [4]

С 1965 года начинается **второе поколение** развития гипертекста. Именно тогда американским философом, социологом (по первому образованию) и первооткрывателем в области информационных технологий *Тедом Нельсоном* (англ. *Ted Nelson*) был **введен в научное обращение термин «гипертекст»** для описания документов, которые выражают нелинейную структуру идей в противоположность линейной структуре традиционных книг, фильмов и речи.

В период 1960–1965 гг. Нельсон занимался разработкой текстового редактора на низкоуровневом языке ассемблера (англ. *assembly language*). В докладе о редакторе на ежегодной конференции «Association of Computing Machinery» (1965 г.) Тед Нельсон определил понятие «гипертекст» следующим образом: *«Гипертекст — это непоследовательный способ записи, текст ветвится, и это позволяет читателю осуществлять выбор и перемещаться по нему с помощью интерактивного экрана».*

Более поздний термин «гипермедиа» близок к определению Нельсона по смыслу, но он подчеркивает наличие в гипертексте нетекстовых компонентов, таких как анимация, записанный звук и видео.

В начале 60-х годов XX в. группа программистов во главе с Тедом Нельсоном начала работу над **проектом «Xanadu»** [5], в котором воплощалась идея гипертекста. Нужно понимать, что эта идея появилась задолго до того, как была реализована первая система гипертекстовых связей в документах. Модель работы Xanadu (рис. 1.6) определяла следующее: каждый документ имеет оригинальную форму (виртуальный файл), но поддерживает видоизменение с версионностью, редактирование и бесконечное дополнение с индексированием.

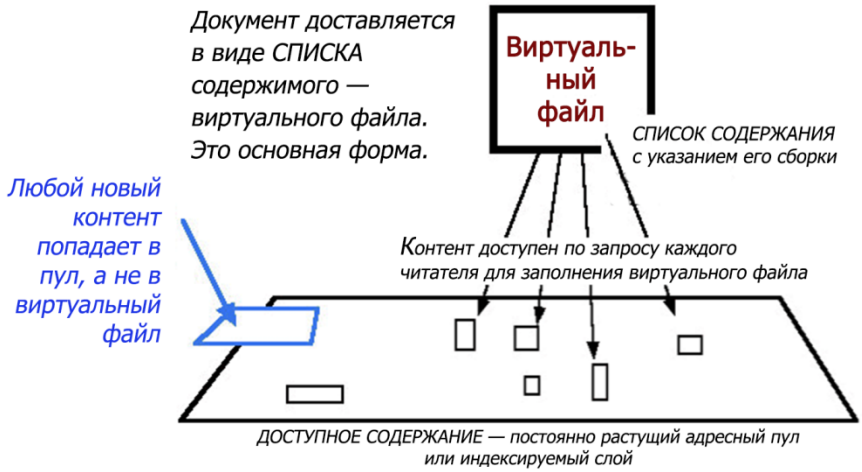


Рис. 1.6. Модель документа Xanadu построена на предположении о постоянном изменении и повторном использовании

Проект Xanadu начался с появлением интерактивных экранов (рис. 1.7) — с предвидения новой «экранной литературы» из параллельных, взаимосвязанных документов. В приведенном на рисунке 1.6 крупном плане видна взаимосвязь между двумя окнами экрана: таким образом автор может показать точную связь между произведениями на разных страницах. Видимая связь

становится структурной частью письма, такой же фундаментальной, как абзац или заголовок. Это обеспечивает обобщенный способ представления параллельных документов и их точных связей.

Разработчики Xanadu закончили работу после 54 лет разработки! Окончательная версия проекта с обновленным названием *«OpenXanadu»* была представлена в апреле 2014 года в Чепменском университете (Калифорния) [6].

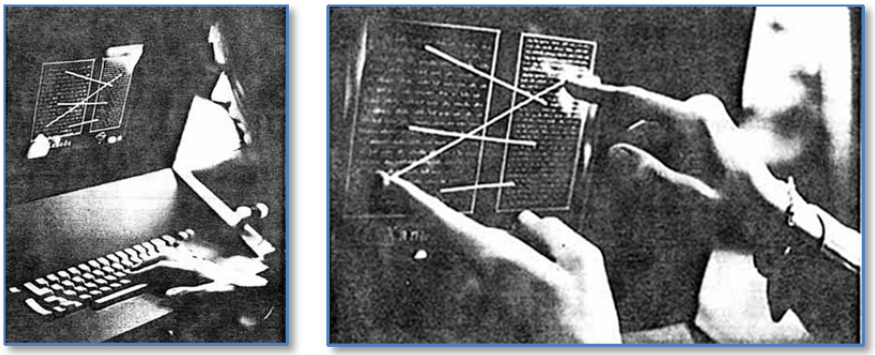


Рис. 1.7. Тед Нельсон сконструировал два экранных окна, соединенных видимыми линиями, показывающими как часть объектов одного окна ссылается на часть объектов другого

Также ко второму поколению развития гипертекста относится целый ряд исследовательских коллективов, занимающихся гипертекстом. Одна из наиболее успешных групп существовала при *американском Университете Брауна* (англ. *Brown University*, штат *Род-Айленд*) в научно-исследовательском институте информационной и преподавательской деятельности (*Institute for Research in Information and Scholarship*, IRIS). В рамках этой группы был создан **проект Intermedia**, руководителем и главным архитектором которого был содиректор института *Норман Мейровиц* (*Norman Meyrowitz*). Проект был рассчитан на двадцатилетнюю работу и опирался на опыт, извлеченный из разработки предшествующих гипертекстовых систем:

1) Гипертекстовая система редактирования Hypertext Editing System (HES), разработанная в 1967 году *Тедом Нельсоном* и несколькими студентами Брауновского Университета (рис. 1.8). Хьюстонский Центр космических полетов с человеком на борту использовал эту систему для подготовки документации по программе «Аполлон».

Рис. 1.8. HES-консоль, Университет Брауна, 1969 г. [9]

2) Система поиска записей и редактирования File Retrieval and Editing System (FRESS), разработанная в 1968 году в Университете Брауна американским профессором информатики *Андрисом ван Дамом* (*Andries van Dam*) и его студентами (уже без участия Теда Нельсона). В начале 70-х годов *концерном Philips* был сделан ее коммерческий вариант. Более 10 лет сотни преподавателей и студентов использовали эту систему для обучения (например, учебный класс английской поэзии вел чтение и записи целиком на гипертекстовом документе, находившемся в общем пользовании). Система FRESS была много-

пользовательской, предусматривала динамическую иерархию и двунаправленные связи, а также узлы и связи с выделением ключевых слов. На графических терминалах поддерживались многооконные изображения и векторная графика [7].

3) Электронная документационная система Electronic Document System (EDS) — гипертекстовая система с акцентом на цветную растровую графику и механизмы навигации (рис. 1.9). EDS — это исследовательский проект американских ученых *Стивена Фейнера* (*Steven K. Feiner*), *Сандора Наги* (*Sandor Nagy*) и *Андриса ван Дама* в 1978–1981 гг. в Университете Брауна [8].

Рис. 1.9. Система EDS – концепция электронного руководства по техническому обслуживанию (1980 г.). Программное обеспечение фактически работало на специальном растровом дисплее VAX 11/780 [11]

Система Intermedia [10] — часть всеобъемлющих усилий, предпринимавшихся в Брауновском Университете с целью повышения эффективности применения рабочих станций в учеб-

ных аудиториях. В этом контексте она разрабатывалась как некоторый комплекс инструментальных средств, позволяющих авторам-пользователям устанавливать связи с текстами, динамическими рядами (*timelines*), диаграммами и другими изображениями, генерируемыми компьютером, а также с документальными видеофильмами и музыкальными записями. Система поддерживала двунаправленные связи для текста и графики. Маленькие иконки использовались для маркировки точек их прикрепления (якорьков – anchors). Связи хранились отдельно от содержимого базы данных.

Система Intermedia разрабатывалась и как средство, позволяющее преподавателям Университета Брауна организовывать и представлять свой учебный материал на компьютере, и как интерактивная среда для студентов, работая в которой они могли изучать этот материал, а также добавлять к нему свой собственный в виде аннотаций или сообщений.

В 1975 году идея гипертекста нашла воплощение в **информационной системе внутреннего распорядка** в качестве ключевого интерфейса между пользователями и материально-техническим обеспечением *атомного авианосца «Карл Винстон»* (*USS Carl Vinson*, *CVN-70*), которая получила название **ZOG** [12]. ZOG была гипертекстовой системой, ранее разработанной в американском Университете Карнеги-Меллона (*Carnegie Mellon University* (*CMU*), *Питтсбург*, штат *Пенсильвания*) в 1970-х годах *Дональдом Маккракеном* (*Donald McCracken*) и *Робертом Акссином* (*Robert Akscyn*), чтобы служить в качестве интерфейса для программ искусственного интеллекта и когнитивных наук, собранных в CMU на летнем семинаре.

ZOG состоял из «фреймов», содержащих заголовок, описание, строку, содержащую системные команды ZOG, и выбор (пункты меню), которые вели к другим фреймам. ZOG впервые применил «рамочную» или «карточную» модель гипертекста, позже популяризированную в HyperCard фирмы Apple. В таких системах рамки или карточки не могут прокручиваться для

отображения содержимого, которое является частью того же документа, но находится за пределами экрана. Вместо этого текст, размер которого превышает размер одного экрана, должен быть помещен на другой (который затем составляет отдельный фрейм или карточку).

Позже проект ZOG стал коммерческим продуктом для долгосрочных исследований искусственного интеллекта, проводимых Алленом Ньюэллом (*Allen Newell*) и финансируемых Управлением военно-морских исследований. В коммерческом варианте эта система получила название **«Система управления знаниями» — KMS** (*Knowledge Management System*).

В **третьем поколении** развития гипертекста появлялись реализации типа HyperCard компании Apple или NodeCards компании Xerox.

Один из ключевых разработчиков программного обеспечения для компьютеров Apple Macintosh, *Билл Аткинсон* (*Bill Atkinson*), вдохновленный идеями Ванневара Буша и его проектом Xanadu, внес большой вклад в популяризацию гипертекста, создав в 1987 году **программу HyperCard**. Это, по сути, визуальная среда программирования, позволяющая создавать собственные приложения гипермедиа на компьютере Macintosh в системе Mac OS (рис. 1.10). С точки зрения пользователя, HyperCard выполняет функцию «организатора информации», позволяющего собирать воедино тексты, картинки, звуки, анимацию и предлагая их пользователю в удобном интерактивном виде.

Основной информационной единицей в HyperCard был компьютерный аналог карточки хранения информации в базе данных. Несколько карточек, объединенных по какому-то принципу, составляют так называемый *информационный стек* (от англ. *stack* — куча, груда, хранилище), который может ассоциироваться с гипертекстовым документом или книгой. При этом принципам гипертекста в HyperCard обладает возможность создания в любом месте информационной карточки специального значка (иконки), отсылающего к тексту другой карточки. Стеки

HyperCard содержат различную информацию (изображение, текст, элемент интерфейса, звук, видео, анимация и пр.). Элементы интерфейса может легко нарисовать сам пользователь или их можно выбрать из меню. Как между карточками одного стека, так и между карточками разных стеков пользователь может создавать связи. Из средств навигационной помощи предоставляются только запоминание пути (история) и поисковые средства.

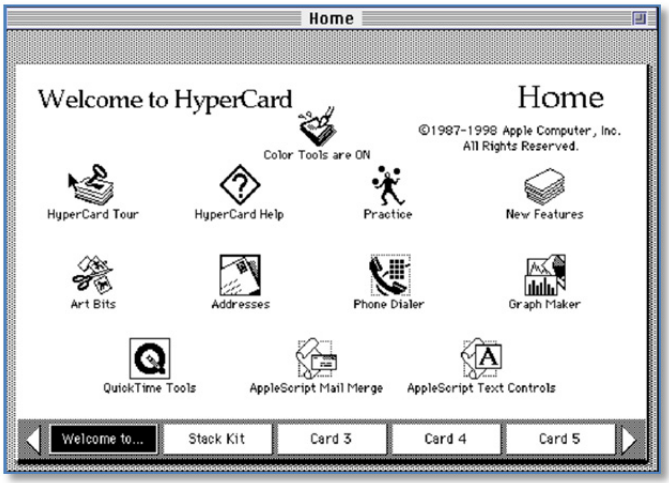


Рис. 1.10. Домашняя страница HyperCard

В HyperCard использовался **язык сценариев HyperTalk**, с помощью которого пользователь мог создавать как сами связи между карточками, так и огромное число дополнительных функций (рис. 1.11). Также HyperTalk позволял без большого труда адаптировать систему для своих нужд.

Сам принцип хранения информации в виде стека требовал, чтобы на каждой карточке присутствовали как минимум *две функциональные кнопки-иконки* — перехода к следующей карточке и возврата назад, к предыдущей карточке. Простота и легкость формирования связей между карточками в системе HyperCard способствовали тому, что иконку-ссылку можно бы-

ло установить в любом месте карточки и связать эту карточку с любой другой в соответствии с намерением и желанием пользователя. Поэтому именно систему HyperCard принято считать первой компьютерной гипертекстовой системой, в которой была реализована возможность оперативно формировать контекстно зависимые ссылки, что впоследствии стало *основой для разработки гипертекстов* и позволило с помощью такой адаптации расширять возможности будущих программ, использующих принцип гипертекста [2, 13].

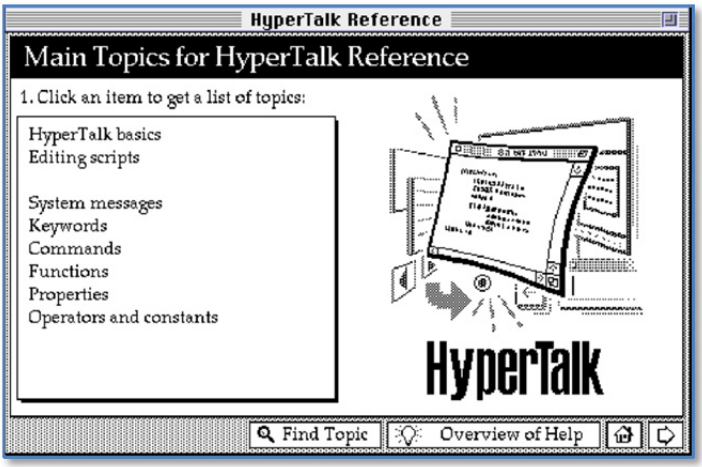


Рис. 1.11. Главное окно для справки по HyperTalk

Программа HyperCard была включена в системное ПО Macintosh и требовала всего 1 Мб оперативной памяти. HyperCard добилась успеха практически мгновенно — в первый год количество проданных копий превысило 1 млн. Окончательный релиз HyperCard 2.4.1 был выпущен в 1998 году.

В 1983 г. увидела свет написанная на языке программирования Lisp **программа ведения заметок с применением перекрестных ссылок NoteCards**, разработанная компанией Xerox. NoteCard обладала «прокручиваемыми» окнами, возможностью работы с единой системой форматирования для всех заметок, а

также *многооконной системой*, в которой просмотрщику заметок (прообраз будущих браузеров) было выделено собственное окно.

В 1987 г. была проведена первая **специализированная конференция Hypertext'87**, материалам которой был посвящен специальный выпуск журнала «Communication ACM».

К 1989 году гипертекст представлял новую, многообещающую технологию, которая имела относительно большое число реализаций с одной стороны, а с другой стороны делались попытки построить формальные модели гипертекстовых систем, которые носили скорее описательный характер и были навеяны успехом реляционного подхода описания данных.

1.1.3. Развитие языка гипертекстовой разметки HTML

В 1969 году сотрудник компании IBM *Чарльз Голдфарб* (*Charles Goldfarb*) возглавил проектирование компьютерной системы обслуживания юридических контор. Под его руководством был создан **первый язык разметки документов Generalized Markup Language** (**GML**), в котором была реализована концепция типа документа (формально определенного шаблона, описывающего схему внутреннего построения схожих документов) и вложенных друг в друга структур. GML не зависел ни от марки компьютеров, ни от операционной системы, и IBM удалось перевести 90 % своей документации в этот формат.

К 1978 г. *комитет по обработке информации Американского национального института стандартов* (англ. *American national standards institute*, ANSI) всерьез заинтересовался языками подготовки гипертекстовых данных. Чарльз Голдфарб возглавил комитет по новому направлению, связанному с формированием стандарта для мощного метаязыка разметки документов, который был назван **стандартным обобщенным языком разметки SGML** (англ. *Standard General Markup Language*), в основу которого был заложен GML. Первый рабочий вариант спецификации появился в 1980 году, а в 1983 г. Ассоциация GCA (англ.

Global Citizens Association) приняла шестую рабочую версию SGML в качестве промышленного стандарта (GCA 101-1983), поддержанного Министерством обороны и Налоговым управлением США. Спустя два года, в 1985 г., сформировалась международная группа пользователей SGML. Считается, что начиная с этого года, человечество вступило в новую эру — развитие современных гипертекстовых технологий.

В 1986 г. международная организация по стандартизации ISO (англ. *International Organization for Standardization*) одобрила **стандарт SGML** (ISO-8879). Он позволил отказаться от конкретных способов представления информации и сосредоточить усилия на продумывании структуры документов с помощью правил определения собственных тегов форматирования, их атрибутов и синтаксиса использования.

SGML оказался очень мощным и универсальным языком, так как требовал точного описания всех нюансов создаваемого синтаксиса документа и подробных правил формирования тегов. В рамках SGML была разработана **концепция DTD** (англ. *Document Type Definition* — определение типа документа) которая позволила связать конкретные *синтаксические правила разбора с заданными способами организации структуры документов*. Взяв на вооружение новую концепцию DTD, многие компании приступили к активной разработке программ анализа SGML-текстов.

Наступил 1989 год, который вошел в историю как **год появления идеи Всемирной паутины** (англ. *World Wide Web*, **WWW**)! Эта идея была предложена (и реализована всего через два года) ***Тимоти Джоном Бернерсом-Ли*** (*Timothy John «Tim» Berners-Lee*) (рис. 1.12), программистом *Европейского центра ядерных исследований* (фр. *Conseil Européen pour la Recherche Nucléaire*, CERN) — крупнейшей в мире лабораторией физики высоких энергий, расположенной в Женеве.

Родители Тима Бернерса-Ли были математиками и участвовали в создании «Manchester Mark I» — одного из первых ком-

пьютеров. Обучаясь в Королевском колледже в Оксфорде, Тимоти самостоятельно собрал свой первый компьютер на базе процессора M6800. Работа в CERN также была отмечена группой проектов, в которых Тим принимал активное участие: создание программы «Enquire» — «Дознаватель» (1980 г.), разработка распределенных систем для сбора научных данных (1984 г.), разработка внутренней системы обмена документов для ENQUIRE (1989 г.). В результате работы над системой обмена документов был предложен глобальный гипертекстовый проект — новое SGML-приложение, известное сегодня как **язык гипертекстовой разметки HTML**.

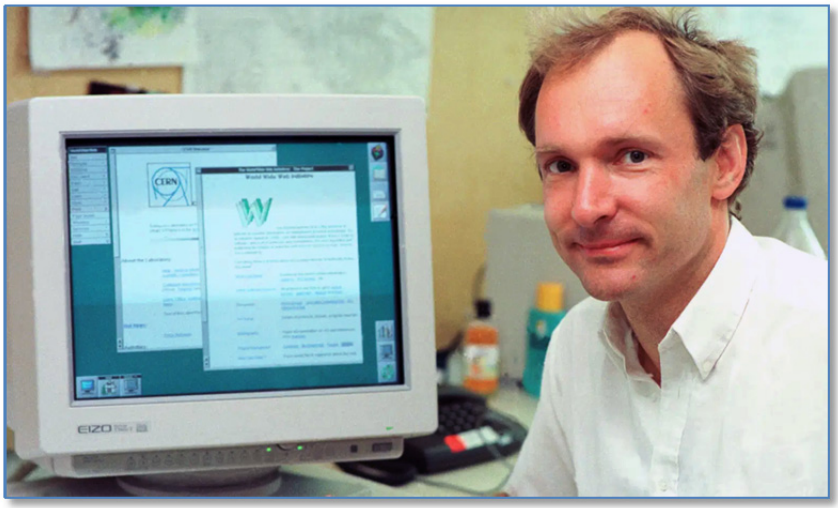


Рис. 1.12. Сэр[1] Тимоти Джон Бернерс-Ли (1992 г.). Создатель URI, URL, HTTP, HTML и Всемирной паутины. Действующий глава Консорциума Всемирной паутины. Автор концепции семантической паутины и множества других разработок в области информационных технологий

[1] В пятницу, 16 июля 2004 года, 49-летний Тим Бернерс-Ли был посвящен Королевой Британии Елизаветой II в Командоры Ордена Британской империи за вклад в глобальное развитие Интернета. Церемония посвящения состоялась в полдень в Букингемском дворце. Его полный титул теперь гласит: «Сэр Тимоти Бернерс-Ли, рыцарь-командор».

В 1990 году Тим Бернерс-Ли написал на языке программирования Objective-C **первый браузер для компьютера NeXT** (рис. 1.13). Браузер включал в себя также первый редактор HTML-документов. Не отставало и наше государство (тогда еще не отставало…) — в декабре 1990 г. при Министерстве связи СССР был открыт **Научно-технический центр гиперинформационных технологий**, известный как ГНТЦ «Гинтех».

Бернерс-Ли начал с философии, согласно которой большая часть академической информации должна быть бесплатно доступна для всех. Он разрабатывал Всемирную паутину как средство для обеспечения доступа к информации для разрозненных по всему миру академических групп через сеть Интернет. Он объединил методы поиска информации и гипертекста, чтобы создать простую, но мощную глобальную информационную систему.

Рис. 1.13. Исторический компьютер NeXT, которым пользовался Тим Бернерс-Ли в 1990 году (в настоящее время экспонат выставки Microcosm в CERN). Это был первый веб-сервер «httpd», гипермедиа-браузер и веб-редактор. Поврежденная этикетка на кубическом системном блоке гласит: «Эта машина является сервером – НЕ ВЫКЛЮЧАЙТЕ ЕГО!!»

Первый в мире сайт был запущен **6 августа 1991 года** по адресу **http://info.cern.ch** (он и сейчас доступен в Сети). На сайте описывается, что такое Всемирная паутина, как установить веб-сервер, как получить и установить свободно распространяемые браузер, веб-редактор и т. п. Этот сайт являлся также первым в мире интернет-каталогом.

В то же время Тим Бернерс-Ли, работая на компьютере NeXT, завершил **первую версию гипертекстового браузера «WorldWideWeb»**. Эта программа является предшественником большей части того, что мы сегодня называем или знаем как «Интернет». В то время как Интернет в основном рассматривается как среда для потребления информации, браузер WorldWideWeb позволял пользователю редактировать и создавать веб-страницы. Это отражает философию Интернета как среды чтения и записи, что хорошо отражено в наблюдении Тима Бернерса-Ли: *«Если вы думаете, что серфинг по гипертексту — это круто, это потому, что вы не пробовали его писать»* [14].

Чтобы сделать Интернет более доступным (потому что почти ни у кого не было машины NeXT), в CERN был разработан второй проект браузера: **браузер линейного режима «Line Mode»**. Браузер был впервые выпущен в 1991 году и был совместим с большинством систем Unix / Linux.

В 1993 г. программист *Роберт Каилау* (*Robert Kailau*), коллега Бернса Ли, сделал **браузер «Samba»** для компьютеров Mac. В том же году сотрудник Института правовой информации (*Legal Information Institute*, LII) *Томас Брюс* (*Thomas R. Bruce*) распространил первый **браузер «Cello»** для компьютеров класса IBM PC, после чего множество компаний начали выпускать собственные Интернет-навигаторы [15]. Это привело к массовой нестыковке, потому что каждый производитель браузеров ста-

рался дополнить тогда еще крайне ограниченный HTML своим набором тегов.

История развития браузеров, в том числе и двух «браузерных войн», широко раскрыта в первой книге «Интернет-технологии для студентов и преподавателей» [16].

Ведущие информационные корпорации, недовольные неразберихой тегов HTML, сформировали в декабре 1994 года **Консорциум Всемирной паутины** (*World Wide Web Consortium*, **W3C**), взявший под свой контроль *работу практически над всеми стандартами важнейших технологий Сети*. Надо отметить, что формально W3C выпускает только рекомендации и некоторые компании их игнорируют, но в целом рекомендации W3C признаются всем IT-рынком в качестве стандартов. В настоящее время главой Консорциума является сэр Тимоти Бернерс-Ли. Сейчас в Консорциум входят более 400 компаний-участников: Adobe, AOL, Apple, Canon, CERN, Cisco, Dow Jones, Google, IBM, Intel, Microsoft, Mozilla, Nokia, Opera, Oracle, Samsung, Siemens, Yahoo и т. д. (полный список участников Консорциума приведен в [17]). Все участники Консорциума занимаются разработкой и продвижением открытых стандартов и рекомендаций для нескольких десятков технологий: HTML, XHTML, CSS, HTTP, URI, XML, DOM, MathML, PNG, SVG, XSLT и пр.

На рис. 1.14 приведена хронология развития языка HTML. Первая версия языка (HTML 1.0) была направлена на представление языка как такового, где описание его возможностей носило скорее рекомендательный характер.

Вторая версия языка (HTML 2.0) фиксировала практику использования его конструкций. Эта версия предоставляла новые возможности, расширяя набор тегов HTML в сторону отображения научной информации и таблиц, а также улучшения стиля компоновки изображений и текста.

Версия 3.2 смогла упорядочить все нововведения и согласовать их с существующей практикой. HTML 3.2 позволяет реали-

зовать использование таблиц, выполнение кодов языка Java, обтекание графики текстом, а также отображение верхних и нижних индексов.

Версия 4.01 включала дополнительные средства работы с мультимедиа, языки программирования, таблицы стилей, упрощенные средства печати изображений и документов. Для управления сценариями просмотра страниц сайта можно было использовать языки программирования сценариев, например, JavaScript.

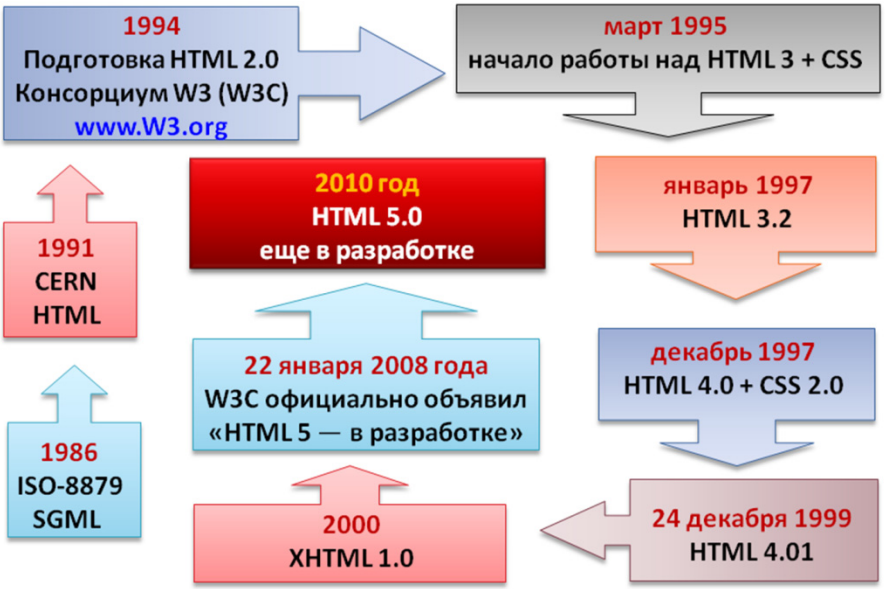


Рис. 1.14. Хронология развития версий языка HTML

Версия 5 вводит несколько новых элементов и атрибутов, которые отражают типичное использование разметки на современных сайтах. Некоторые устаревшие элементы, которые можно было использовать в HTML 4.01, были исключены, включая чисто оформительские элементы, такие как `<font>` и `<center>`, чьи эффекты выполняются с помощью каскадных таблиц стилей CSS. Синтаксис HTML5 больше не базируется на SGML, не-

смотря на подобие его разметки. Также получили развитие языки, разрабатывающиеся параллельно с HTML: VRML (англ. *Virtual Reality Modelling Language* — язык моделирования виртуальной реальности) и XML (англ. *eXtensible Markup Language* — язык расширенной разметки).

Версия HTML5 до сих пор еще не получила статус официальных рекомендаций W3C, но уже сейчас понятно, что авторы HTML продолжают работать в направлении разработки требований к поддержке объектной модели документа и интерпретации языка JavaScript.

Язык HTML, на котором написано большинство веб-страниц, позволяет передавать размеченное содержание от источника к получателю в очень компактной форме. Такой принцип организации, а именно разделение формы и содержания веб-страниц, лежит в основе связки HTML + CSS. В частности, Д. Кирсанов в книге «Web-дизайн» описывает особенности HTML следующим образом: *«Этот обобщенный метаязык предназначен для построения систем логической, структурной разметки любых разновидностей текстов. Слово «структурная» означает, что управляющие коды, вносимые в текст при такой разметке, не несут никакой информации о форматировании документа, а лишь указывают границы и соподчинение его составных частей, т. е. задают его структуру»* [18, с. 19–20].

Современный Интернет — это постоянно растущее число связанных между собой веб-страниц и веб-приложений. Взаимодействие веб-технологий и браузеров представляет собой взаимное переплетение которое удачно представлено авторами проекта «EvolutionOfWeb» [19]. Современный Интернет является результатом непрерывных усилий открытого веб-сообщества, которое помогает разрабатывать такие технологии, как HTML5, CSS3 и WebGL, добивается их поддержки всеми браузерами.

Цветные ленты на инфографике (рис. 1.15–1.18) представляют взаимосвязи между веб-технологиями и браузерами в период 1991–2013 гг., благодаря которым пользователю доступно огромное множество функциональных веб-приложений.

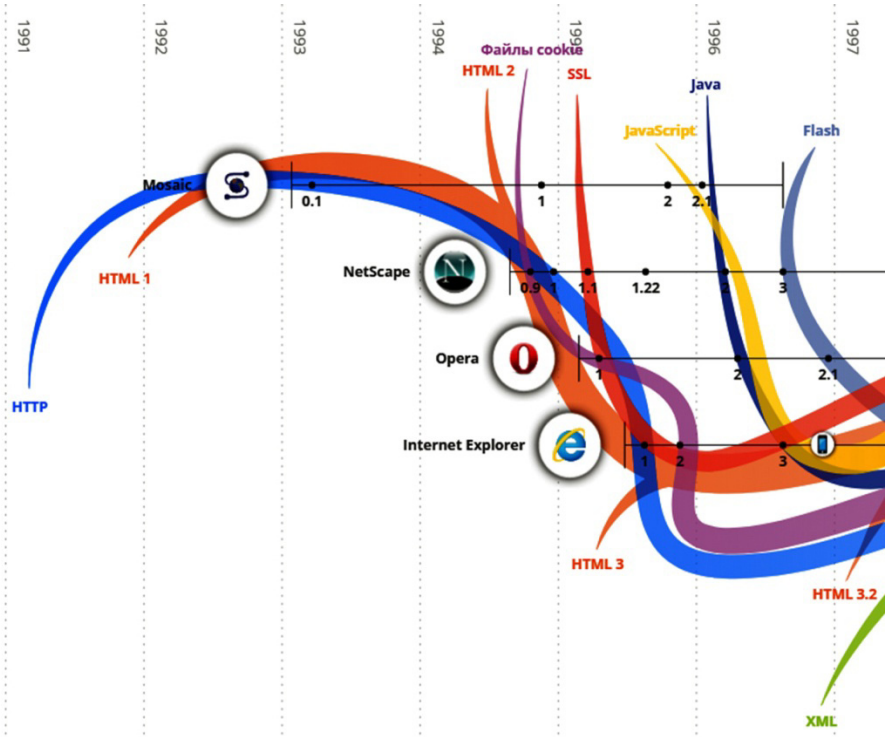


Рис. 1.15. Эволюция Интернета. Браузеры и технологии. 1991–1997 гг. [19]

1.2. Язык гипертекстовой разметки

Разработчикам языка HTML удалось решить две ключевые задачи:

- предоставить дизайнерам гипертекстовых баз данных простое средство создания документов;
- сделать это средство достаточно мощным, чтобы отразить имевшиеся на тот момент представления об интерфейсе пользователя гипертекстовых баз данных.

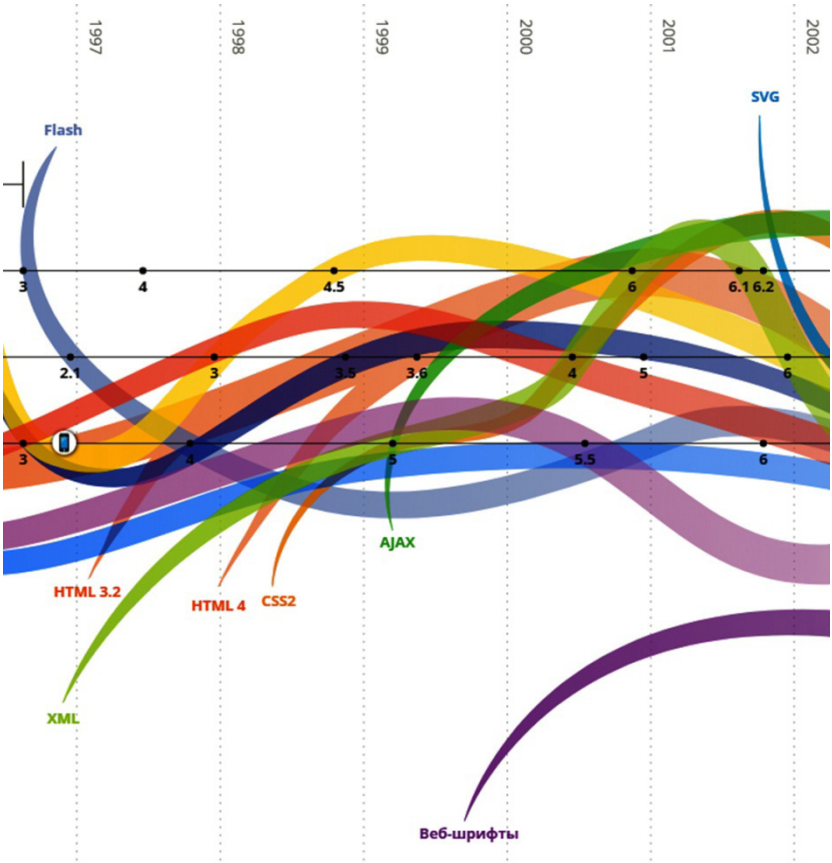


Рис. 1.16. Эволюция Интернета.
Браузеры и технологии. 1997–2002 гг. [19]

Первая задача была решена за счет **выбора теговой модели описания документа**. Такая модель широко применяется в системах подготовки документов для печати.

Язык HTML позволяет размечать электронный документ, который может содержать самые разнообразные метки, иллюстрации, аудио- и видеофрагменты и т. д. В состав языка вошли развитые средства для создания различных уровней заголовков, шрифтовых выделений, различные списки, таблицы, текстовые блоки и многое другое.

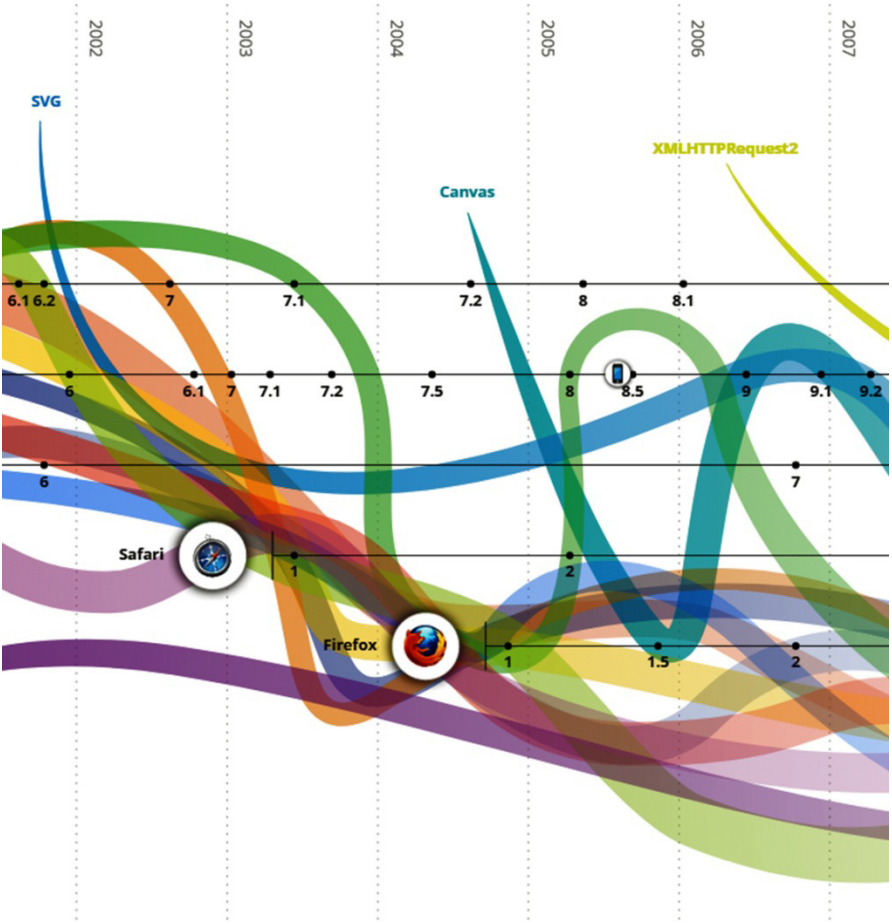


Рис. 1.17. Эволюция Интернета.
Браузеры и технологии. 2002–2007 гг. [19]

Вторым важным моментом, повлиявшим на судьбу HTML, стало то, что **в качестве основы был выбран обычный текстовый файл**.

Таким образом, гипертекстовая база данных в концепции WWW — это набор текстовых файлов, размеченных на языке HTML, который определяет форму представления информации (разметка) и структуру связей между этими файлами и другими информационными ресурсами (гипертекстовые ссылки).

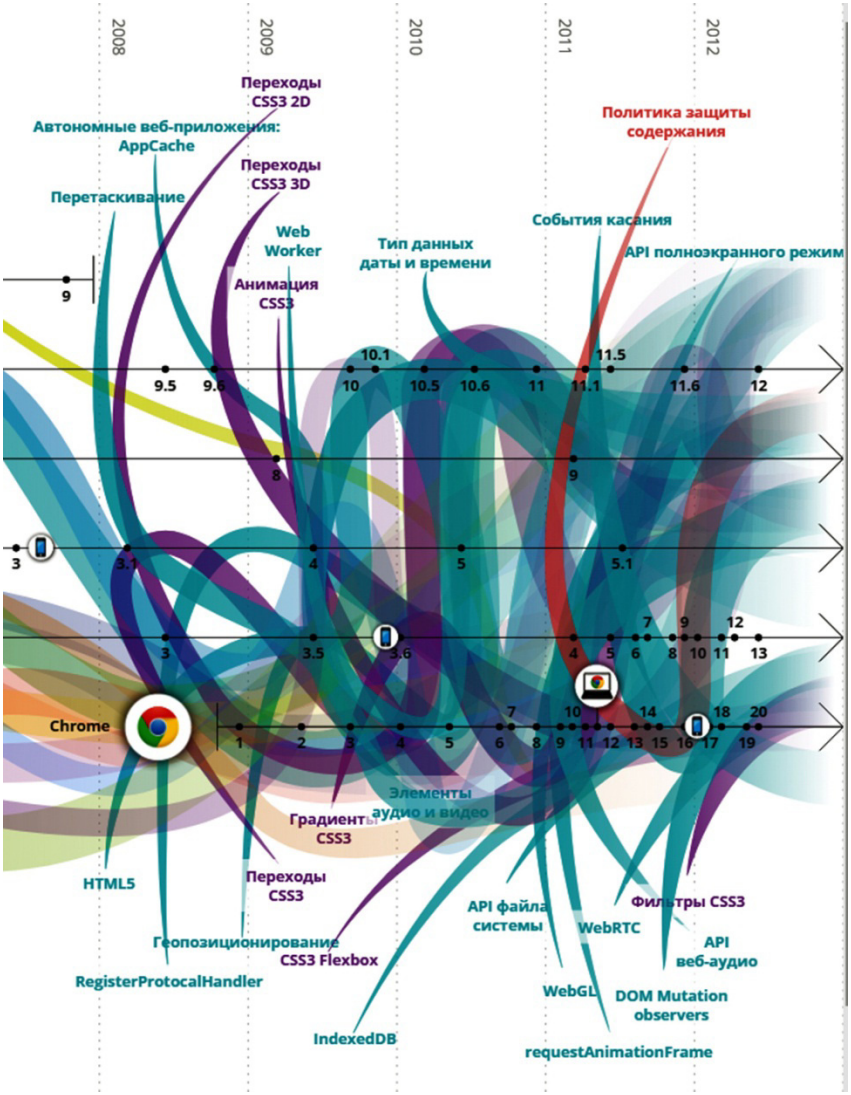


Рис. 1.18. Эволюция Интернета. Браузеры и технологии. 2007–2013 гг. [19]

Гипертекстовые ссылки, устанавливающие связи между текстовыми документами, постепенно стали объединять самые различные информационные ресурсы, в том числе звук и видео (гипермедиа).

Такой подход предполагает наличие еще одного компонента технологии — интерпретатора языка. В World Wide Web функции интерпретатора разделены между веб-сервером гипертекстовой базы данных и интерфейсом пользователя (модель *«Клиент-Сервер»*, см. п. 1.2.2). Сервер, кроме доступа к документам и обработки гипертекстовых ссылок, обеспечивает предпроцессорную обработку документов, в то время как интерфейс пользователя осуществляет интерпретацию конструкций языка, связанных с представлением информации.

1.2.1. Принципы теговой модели

Как было отмечено ранее, язык HTML является приложением («частным случаем») SGML (англ. *Standard Generalized Markup Language* — стандартный обобщенный язык разметки) и соответствует международному стандарту ISO 8879.

HTML (от англ. ***H**yper**T**ext **M**arkup **L**anguage — язык гипертекстовой разметки*) — стандартный язык разметки документов во Всемирной паутине.

Подавляющее большинство веб-страниц содержат описание разметки на языке HTML (или XHTML[1]). Язык HTML интер-

[1] XHTML (англ. *eXtensible HyperText Markup Language* — расширяемый язык гипертекстовой разметки) — семейство языков разметки веб-страниц на основе XML, повторяющих и расширяющих возможности HTML 4.
В конце 90-х годов быстро набрал популярность новый язык разметки — XML. Он представляет собой упрощенный вариант SGML и также позволяет создавать пользовательские XML-приложения.
В январе 2000 г. вышла спецификация нового языка разметки XHTML 1.0, в которую консорциум W3C предложил переносить существующие HTML-материалы. Она была названа «переформулировкой HTML 4.0 в виде прило-

претируется браузерами и отображается в виде документа в удобной для человека форме.

В HTML используется теговая модель описания документа как совокупности элементов, каждый из которых окружен тегами.

> **Элемент** — конструкция языка HTML, предписывающая способ интерпретации помещенных внутри нее данных.
>
> **Тег** — единица разметки, стартовый или конечный маркер элемента. Теги определяют границы действия элементов.
>
> **Атрибут** — параметр или свойство тега. Все атрибуты записываются внутри стартового тега и разделяются пробелами.

По своему значению и принципам использования **теги близки к понятию скобок begin/end** в языках программирования. В HTML они обрамляются угловыми скобками «<» и «>».

Теги определяют область действия правил интерпретации текстовых элементов документа. Элементы документа, размеченного при помощи HTML, также принято называть **контейнерами**. Общая схема построения элемента в формате HTML приведена на рис. 1.19.

Рис. 1.19. Элемент языка HTML включает в себя открывающий и закрывающий теги, группу атрибутов со значениями внутри открывающего тега и контент – содержимое (как правило, текстовое) элемента, размещенного между тегами

жения XML 1.0» (хотя до этого HTML считался SGML-приложением). Очень важной оказалась возможность создания в XHTML-приложении собственных тегов (поскольку XHTML основывается на XML). Консорциум W3C назвал XHTML технологией, предназначенной для мягкого перехода с HTML на XML.

В общем случае, элемент, маркированный тегом, имеет следующую структуру (имя тега соответствует имени элемента, т. е в данном случае можно говорить о теге `<element>` либо об элементе с именем `element` — в книге будут использоваться оба эти обращения при описании элементов):

```
<element attr1=“value1” attr2=“value2”...>
   внутреннее содержание
</element>
```

Если тег не имеет внутреннего содержания, то он может иметь упрощенную структуру:

```
1  <element attr1=“value1” attr2="value2”...>
2  <element attr1=“value1” attr2=“value2”... />
```

В строке 2 в закрывающем теге использована конструкция «/>», а не просто «>». Это необходимо, если используется версия языка XHTML 1.0 — в этом случае элементы должны закрываться (признак закрывающего тега — косая черта (слеш) перед именем тега), даже если использование закрывающего тега запрещено (естественно, ввиду отсутствия содержимого элемента). При этом во избежание ошибок при обработке документа перед этим сочетанием символов нужно вставить пробел [19].

В языке HTML имеют место элементы, например, `img` и `hr`, в которых запрещен закрывающий тег в виду отсутствия содержимого (контента):

```
<img src=“photo.jpg” alt=“My photo”>
<hr>
```

Теговая модель описывает документ как совокупность контейнеров, *каждый из которых начинается и заканчивается тегами*, то есть html-документ представляет собой не что иное, как обычный ASCII-файл, с добавленными в него управляющими html-кодами (тегами). Поскольку HTML произошел от SGML, в нем разрешено использовать только *три управляющих символа*:

горизонтальную табуляцию, перевод каретки и перевод строки. Это облегчает взаимодействие html-документа (файла) с различными операционными системами.

Теговая модель также определяет **принцип вложенности элементов** с одним правилом: закрывающие теги вложенных элементов должны встречаться в разметке в обратном порядке относительно одноименных открывающих тегов (пересечение / наложение элементов запрещено). Например,

```
1  <div id="photomag">
2     <a href="ivanenko_big.jpg">
3        <img src="photo.jpg" title="Магистр ДонНТУ"
4             alt="Магистр ДонНТУ Иваненко Иван Иванович">
5     </a>
6  </div>
```

В строке 1 начинается элемент `div` (открывающий тег `<div>` с атрибутом `id`). В него вложен элемент `a` (открывающий тег `<a>` с атрибутом `href` в строке 2), в который, в свою очередь вложен элемент `img` (единственный тег `<img>` в строке 3 с атрибутами `src`, `title` и `alt`). Закрытие элементов происходит строго по правилу теговой модели: так как у элемента `img` нет закрывающего тега, то сначала закрывается элемент `a` (строка 5), а потом элемент `div` (строка 6). Если условно убрать все атрибуты, то приведенная выше разметка будет выглядеть следующим образом:

```
1  <div>
2     <a>
3        <img>
4
5     </a>
6  </div>
```

Здесь более наглядно просматривается принцип вложенности, выраженный в очередности закрывающих тегов по отношению к открывающим.

1.2.2. Технология «Клиент-Сервер»

Сразу приведем определение клиент-серверной технологии:

> **«Клиент-сервер»** (англ. *client-server*) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми *серверами*, и заказчиками услуг, называемыми *клиентами*.

Фактически клиент и сервер — это программное обеспечение. Взаимодействие клиента и сервера происходит через компьютерную сеть и производится, как правило, на различных физических устройствах.

Программы-серверы ожидают от программ-клиентов запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством протоколов HTTP, FTP, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр веб-страниц). Как правило, одна программа-сервер выполняет запросы от множества программ-клиентов, ее размещают на специально выделенной *вычислительной машине* (ВМ) с высокой производительностью. Из-за особой роли такой ВМ, ее также называют сервером, а ВМ, выполняющие клиентские программы, соответственно, клиентами.

На рис. 1.20 приведена схема клиент-серверной технологии. Со стороны клиентской части, в качестве инициатора HTTP-запросов к серверу, выступает браузер, обладающий модулями для обработки поступающих html-документов, а также способного выполнить сценарии JavaScript и обработать стили CSS (этими качествами обладают все без исключения современные браузеры). При поступлении запроса, веб-сервер начинает его обрабатывать путем формирования ответа (в большинстве случаев это html-документы) обращаясь к внутренним ресурсам (серверные модули для «сборки» динамических веб-страниц,

базы данных, вспомогательные приложения и т. д.). Таким образом, эта достаточно простая технология постоянно работает в режиме «запрос — ответ».

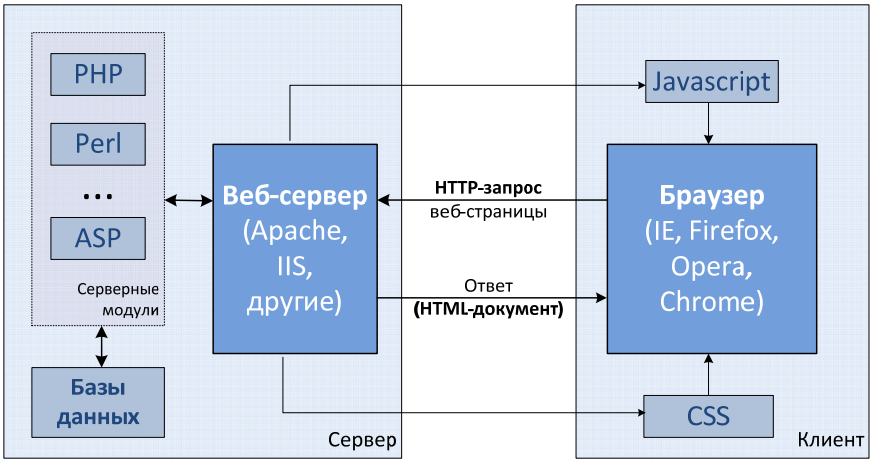


Рис. 1.20. Клиент-серверная технология

Преимущества технологии «Клиент-сервер»:

– в большинстве случаев возможно распределение вычислительной системы между несколькими независимыми ВМ;
– повышенная защита сервера, на котором хранятся данные, по отношению к защите большинства ВМ клиентов.
– простота организации контроля полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

Недостатки технологии «Клиент-сервер»:

– неработоспособность сервера (отказ от обслуживания ввиду недостатка производительности, либо физический износ) способна сделать неработоспособной всю вычислительную сеть;
– поддержка работы данной системы требует отдельного специалиста — системного администратора;
– высокая стоимость сетевого оборудования.

1.2.3. Обработка веб-документов в браузере

Основное предназначение браузера — отображение веб-документов. Для этого, согласно технологии «Клиент-сервер», от клиента на сервер отправляется запрос, а результат выводится в окне браузера.

Вывод запрошенного содержимого на экране монитора компьютера клиента осуществляется с помощью специального **модуля отображения** в браузере (рис. 1.21). По умолчанию он способен отображать html- и xml-документы, а также графические файлы. Специальные подключаемые модули (расширения для браузеров) делают возможным отображение другого содержания, например PDF-файлов.

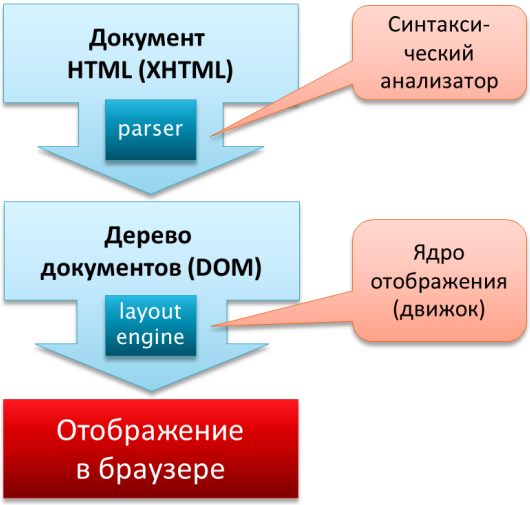


Рис. 1.21. Модуль отображения веб-документов в браузере

Модуль отображения выполняет **синтаксический анализ веб-документа** и переводит теги в узлы **объектной модели документа DOM** (от англ. *Document Object Model*). Также извлекается информация о стилях как из внешних css-файлов, так и из одноименных элементов и атрибутов `style` (см. п. 2.2.3). Загру-

женные и распознанные стили формируют **модель CSSOM** (англ. *CSS Object Model*). Эта информация и инструкции по отображению в html-файле используются для создания еще одного дерева — **дерева отображения** (дерево рендеринга, или *render tree*). Оно содержит узлы-блоки с визуальными атрибутами, такими как цвет и размер. Узлы-блоки располагаются в том порядке, в каком они должны быть выведены на экран.

После создания дерева отображения начинается **компоновка элементов** (*layout*), в ходе которой каждому узлу присваиваются координаты точки на экране, где он должен появиться. Затем выполняется отрисовка, при которой узлы дерева отображения **последовательно отрисовываются** (браузеры используют поточный метод (*flow*), при котором в большинстве случаев достаточно одного прохода для размещения всех элементов) с помощью исполнительной части пользовательского интерфейса.

Важно понимать, что это последовательный процесс. Для удобства пользователя модуль отображения старается вывести содержание на экран как можно скорее, поэтому создание дерева отображения и компоновка могут начаться еще до завершения синтаксического анализа кода HTML. Одни части документа анализируются и выводятся на экран, в то время как другие только передаются по сети.

В процессе взаимодействия пользователя со страницей, а также выполнения скриптов, последовательность отображения веб-документов в браузере меняется, что требует повторного выполнения некоторых из вышеперечисленных операций. К таким операциям относятся переорисовка и перерасчет положения на странице.

Переотрисовка (*repaint* или *restyle*) происходит в случае изменения стилей произвольного элемента, не влияющих на его размеры и положение на странице (например, `background-color`, `border-color`, `visibility`). При этом браузер просто отрисовывает его заново, с учетом нового стиля.

Перерасчет положения на странице (*reflow* или *relayout*) происходит, если изменения затрагивают содержимое, структу-

ру документа, положение элементов. Причинами таких изменений обычно являются:

- манипуляции с DOM (добавление, удаление, изменение, перестановка элементов);
- изменение содержимого текста в полях форм;
- расчет или изменение css-свойств;
- добавление, удаление таблиц стилей;
- манипуляции с атрибутом `class` (отработка скриптов);
- манипуляции с окном браузера (изменение размеров, прокрутка страницы);
- активация псевдо-классов (например, `:hover`, см. 2.3.9).

Браузеры по возможности локализуют *repaint* и *reflow* в пределах элементов, подвергнувшимися изменению. Например, изменение размеров абсолютно или фиксировано спозиционированного элемента (см. п. 2.5.2) затронет только сам элемент и его потомков, в то время как изменение статично спозиционированного — повлечет reflow всех элементов, следующих за ним.

Еще одна особенность — во время выполнения JavaScript браузеры сохраняет в кэше[1] вносимые изменения, и применяют их в один проход по завершению работы блока кода.

***

Детально разберемся с упомянутыми выше понятиями. Прежде всего, это модули отображения (ядро или движок браузера), которые, по сути, позиционируются как авторские программные продукты (ноу-хау) браузерных компаний и в большинстве случаев являются коммерческой тайной.

В Firefox применяется модуль **Gecko** — собственная разработка компании Mozilla, а в браузерах Safari и Chrome использу-

[1] Кэш-память (или просто — кэш) браузера содержит все документы, загруженные пользователем по протоколу HTTP. Используется для доступа к ранее загруженным страницам при навигации назад/вперед, позволяет сохранять страницы, или просматривать их код, не обращаясь повторно к серверу. Настройки кэш-памяти (например, очистка кэша) доступны пользователю в настройках браузера.

ется движок **WebKit** [20]. WebKit, совместно разработанный компаниями Apple, Adobe, KDE и др., представляет собой модуль отображения с открытым исходным кодом, который был изначально разработан для платформы Linux и адаптирован компанией Apple для Mac OS и Windows. Кроме Gecko и WebKit, в настоящее время активно разрабатываются следующие движки (в скобках указана компания-разработчик): **Blink** (Google), **Goanna** (Mozilla), **KHTML** (KDE), **Prince** (YesLogic), **Servo** (Mozilla), **EdgeHTML** (Microsoft).

Следующее понятие, являющееся важным этапом работы модуля отображения — синтаксический анализ (англ. *parsing* — парсинг), — преобразование веб-документа в пригодную для чтения и выполнения структуру. Результатом синтаксического анализа, как правило, является дерево узлов, представляющих структуру документа. Оно называется деревом синтаксического анализа, или просто синтаксическим деревом [21].

Синтаксический анализ работает на основе определенных правил, которые определяются языком (форматом) документа. Для каждого формата существуют грамматические правила, состоящие из словаря и синтаксиса. Они образуют т. н. бесконтекстную грамматику, в которой грамматические правила могут быть полностью выражены в формате BNF[1].

Синтаксические анализаторы бывают двух типов: нисходящие и восходящие. Первые выполняют анализ сверху вниз, а

[1] Форма Бэкуса — Наура (англ. *Backus Normal Form*, BNF) — формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. BNF -конструкция определяет правила замены символа (нетерминала) на какую-то последовательность букв (терминалов) и символов. Процесс получения цепочки букв можно определить поэтапно: изначально имеется один символ, затем этот символ заменяется на некоторую последовательность букв и символов, согласно одному из правил. После процесс повторяется и, в конце концов, получается цепочка, состоящая из букв и не содержащая символов [22]. Это означает, что полученная цепочка может быть выведена из начального символа. BNF используется для описания контекстно-свободных формальных грамматик.

вторые — снизу вверх. Нисходящие анализаторы разбирают структуру верхнего уровня и ищут соответствия синтаксическим правилам. Восходящие анализаторы сначала обрабатывают входную последовательность символов и постепенно выявляют в ней синтаксические правила, начиная с правил нижнего и заканчивая правилами верхнего уровня.

Задача синтаксического анализатора HTML — переводить информацию из кода HTML в синтаксическое дерево. Словарь и синтаксис языка HTML определены в спецификациях W3C.

Однако HTML (в отличие от XHTML, CSS и JavaScript) невозможно определить с помощью бесконтекстной грамматики, с которой работают синтаксические анализаторы, поскольку грамматика формального стандарта определения HTML — формата DTD (*Document Type Definition*), — не является бесконтекстной. Это связано с «мягким» синтаксисом HTML — способность языка «прощать» ошибки (вплоть до пропуска открывающих или закрывающих тегов, которые подставляются автоматически) ощутимо облегчает жизнь разработчику. С другой стороны, из-за этого становится сложно формально определить грамматику.

Поэтому определение HTML включено в формат DTD (содержит определения всех допустимых элементов, их атрибутов и иерархии), в котором не задается бесконтекстная грамматика. При этом существует несколько версий DTD (см. п. 1.2.4).

Полученное синтаксическое дерево состоит из элементов DOM и узлов атрибутов.

> **DOM** (от англ. *Document Object Model* — «объектная модель документа») — специальная, языково-независимая интерфейсная модель разбора документов XML и HTML.

DOM также служит для представления html-документа и интерфейса элементов HTML таким внешним объектам, как код JavaScript.

Модель DOM практически идентична разметке. В самой разметке следует различать **символьные данные** (*character data*) и **компоненты разметки** (*markup*). Символьные данные представляют информацию, которая будет отображена в окне браузера, в то время как компоненты разметки (невидимы для конечного пользователя через окно браузера) определяют структуру этого отображения.

Рассмотрим пример разметки. Здесь темно-серым цветом выделены только символьные данные:

```
<p><a href="http://www.w3.org/">Консорциум W3C</a>
Разрабатывает<strong>стандарты</strong> HTML и CSS.</p>
```

Эта же разметка с выделенными компонентами разметки (открывающие и закрывающие теги):

```
<p><a href="http://www.w3.org/">Консорциум W3C</a>
Разрабатывает<strong>стандарты</strong> HTML и CSS.</p>
```

Объектная модель приведенного выше фрагмента кода HTML представлена на рис. 1.22. В ней, согласно разметке, имеют место узлы элемента и текстовые узлы. Между узлами элементов существуют определенные отношения (см. п. 1.3.1).

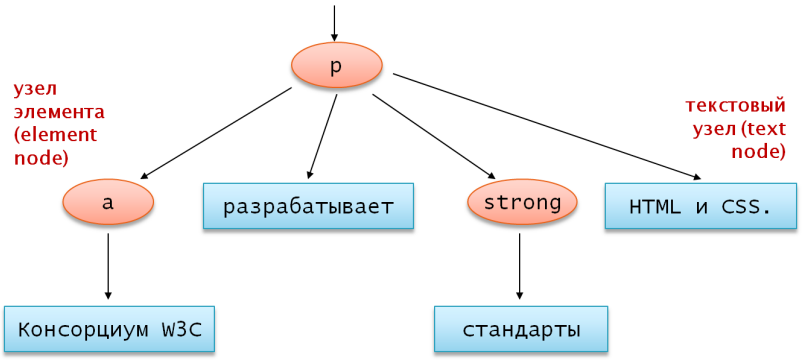


Рис. 1.22. Объектная модель фрагмента кода

1.2.4 Структура документа HTML

Любой гипертекстовый документ HTML состоит из следующих обязательных частей (рис. 1.23):

1. Версия html-документа (**DOCTYPE**).
2. Обозначение языка документа (**HTML**).
3. «Голова» документа (**HEAD**).
4. Заголовок окна веб-страницы (**TITLE**).
5. «Тело» документа (**BODY**).

На рис.1.24 приведен пример разметки с обязательными элементами и построенной объектной моделью документа.

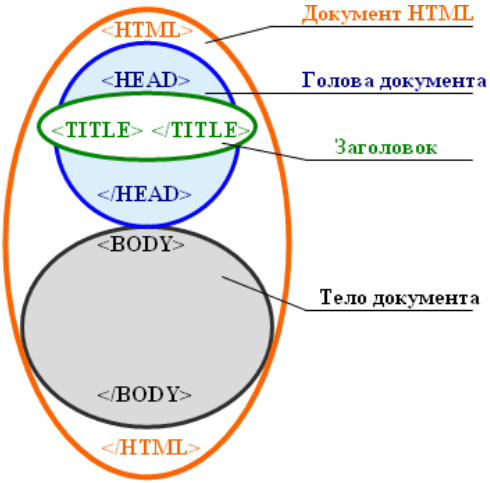


Рис. 1.23. Структура документа HTML

Детально рассмотрим обязательный элемент html-разметки, определяющий тип текущего документа.

Элемент **<!DOCTYPE>** предназначен для указания типа текущего документа — DTD. Это необходимо, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу. Чтобы браузер «не путался» и понимал, согласно какому стан-

дарту отображать веб-страницу, необходимо в первой строке кода прописывать <!DOCTYPE>. При этом <!DOCTYPE> не участвует в построении DOM (см. рис. 1.24).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>My first page</title>
  </head>
  <body>
    <p>Hello, World!</p>
  </body>
</html>
```

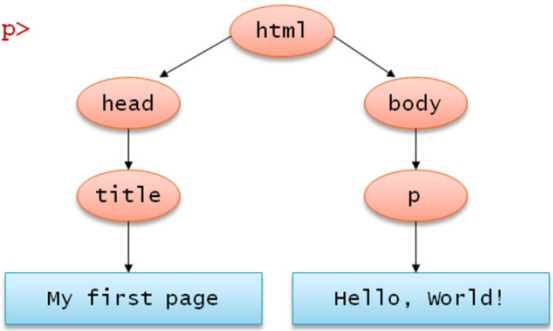


Рис. 1.24. Пример разметки HTML с обязательной структурой и объектной моделью документа

Существует несколько видов <!DOCTYPE>, они различаются в зависимости от версии языка, на который ориентированы. В таблице 1.1 приведены основные типы документов с их описанием.

Синтаксис элемента <!DOCTYPE>:

```
<!DOCTYPE [Элемент верхнего уровня] [Публичность]
  "[Регистрация]//[Организация]//[Тип] [Имя]//[Язык]"
  "[URL]">
```

Синтаксис включает следующие параметры:

✓ *Элемент верхнего уровня* — указывает элемент верхнего уровня в документе, для HTML это тег <html>.

✓ *Публичность* — объект является публичным (значение PUBLIC) или системным ресурсом (значение SYSTEM), например,

таким как локальный файл. Для HTML/XHTML указывается значение `PUBLIC`.

✓ *Регистрация* — сообщает, что разработчик DTD зарегистрирован в международной организации по стандартизации (*International Organization for Standardization*, ISO). Принимает одно из двух значений: плюс «+» — разработчик зарегистрирован в ISO и минус «–» — разработчик не зарегистрирован. Для W3C значение ставится «–».

✓ *Организация* — уникальное название организации, разработавшей DTD. Официально HTML/XHTML публикует `W3C`, это название и пишется в `<!DOCTYPE>`.

Таблица. 1.1 Допустимые форматы DTD

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 документе HTML применяются фреймы
HTML 5	
`<!DOCTYPE html>`	для всех документов
XHTML 1.1	
`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" “http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd”>`	строгий синтаксис XHTML

✓ *Тип* — тип описываемого документа. Для HTML/XHTML указывается значение `DTD`.

✓ *Имя* — уникальное имя документа для описания DTD.

✓ *Язык* — язык, на котором написан текст для описания объекта. Содержит две буквы, пишется в верхнем регистре. Для документа HTML/XHTML указывается английский язык (`EN`).

✓ *URL* — адрес документа с DTD.

При **строгом синтаксисе** HTML, описанном в файле **strict.dtd**, допускается включать все теги и атрибуты, кроме *осуждаемых*. Правила синтаксиса зависят от используемой версии, но в целом должны соблюдаться следующие:

1. Осуждается использование следующих тегов: `<applet>`, `<basefont>`, `<center>`, `<dir>`, `<font>`, `<isindex>`, `<noframes>`, `<plaintext>`, `<s>`, `<strike>`, `<u>`, `<xmp>`. Взамен по возможности рекомендуется использовать стили.

2. Текст, изображения и элементы форм нельзя напрямую добавлять в `<body>`, эти элементы должны обязательно находиться внутри блочных элементов `<p>` или `<div>`.

3. Осуждается применение атрибутов `target`, `start` (тег `<ol>`), `type` (теги `<li>`, `<ol>`, `<ul>`) и др.

Естественно, также должен строго соблюдаться синтаксис языка — правильное вложение тегов, закрытие тегов, должны присутствовать обязательные теги и др.

При **переходном синтаксисе** HTML, описанном в файле **loose.dtd**, применяется «мягкий» синтаксис языка, в котором допускается использовать все теги и атрибуты, включая осуждаемые. Цель переходного `<!DOCTYPE>` заключается в постепенном знакомстве с синтаксисом языка. Если сразу использовать строгий синтаксис начинающему веб-разработчику покажется слишком сложным, для него и предназначен переходный синтаксис HTML.

Синтаксис с фреймами в HTML, описанном в файле **frameset.dtd**, применяется во фреймовой структуре в документе, и аналогичен переходному синтаксису. Данный `<!DOCTYPE>` применяется только для главного документа, формирующего структуру фреймов с помощью тегов `<frameset>` и `<frame>`.

В HTML5 существует лишь один тип `<!DOCTYPE html>`, который переводит браузер в стандартный режим (строгий синтаксис HTML).

1.2.5 Обязательные элементы HTML

Элемент `html` является контейнером, который заключает в себе все содержимое веб-страницы, включая элементы `head` и `body`. Открывающий и закрывающий теги `<html>` и `</html>` в документе необязательны (браузер в состоянии их расставить автоматически, но хороший тон веб-разработки рекомендует всегда расставлять теги, даже там, где они не обязательны, но не запрещены).

Как правило, тег `html` идет в документе вторым, после определения типа документа (DTD), устанавливаемого через элемент `!DOCTYPE`. Закрывающий тег `<html>` должен всегда стоять в документе последним:

HTML

```
<!DOCTYPE html>
<html>
  <head>
        . . .
  </head>
  <body>
        . . .
  </body>
</html>
```

Элемент `html` является корневым при построении модели DOM, т. е. с позиции формирования дерева HTML, из узла элемента `html` выходят только две «ветви» — к узлам элементов `head` и `body`.

🖫 **Элемент head** предназначен для хранения других элементов, цель которых — помочь браузеру в работе с данными.

Внутри контейнера head находятся **метатеги**, которые используются для хранения информации предназначенной браузерам и поисковым системам. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных.

В шаблоне персонального сайта магистров [23] элемент head заполнен так, как показано на рис. 1.25.

```
3
4  <head>
5
6    <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
7
8    <title> Резюме -
9            Иваненко Иван Иванович -
10           Оптимизация аппаратурных затрат в логических
11           схемах устройств управления
12   </title>
13
14   <link rel="stylesheet" type="text/css" href="css/master_style.css">
15
16 </head>
17
```

Кодовая таблица символов

Заголовок документа

Каскадные таблицы стилей

Рис. 1.25. Пример разметки элемента head

Содержимое элемента head не отображается напрямую на веб-странице, за исключением тега <title> устанавливающего заголовок окна веб-страницы.

Внутри контейнера head допускается размещать следующие элементы (некоторые из них будут рассмотрены ниже): base, basefont, bgsound, link, meta, script, style, title.

🖫 **Элемент body** предназначен для хранения **содержания веб-страницы (контента)**, отображаемого в окне браузера. Информация, которую браузер выводит на экран компьютера клиента, располагается именно внутри контейнера <body>. К такой информации относится текст, изображения, теги, скрипты JavaScript и т. д.

Открывающий и закрывающий теги <body> и </body> на веб-странице не являются обязательными, однако хорошим стилем считается их использование, чтобы определить начало и конец документа HTML.

В открывающем теге <body> могут размещаться определенные атрибуты, и обработчики событий, например, `onload` (указатель полностью загруженной веб-страницы). Для элемента body могут быть применены следующие атрибуты:

- ✓ `alink` — устанавливает цвет активной ссылки;
- ✓ `background` — задает фоновое изображение веб-страницы;
- ✓ `bgcolor` — устанавливает цвет фона веб-страницы;
- ✓ `bgproperties` — определяет, прокручивать рисунок фона совместно с текстом (атрибут не используется или ему задается пустое значение — "") или нет (задается значение "`fixed`");
- ✓ `bottommargin` — задает отступ от нижнего края окна браузера до контента;
- ✓ `leftmargin` — задает отступ по горизонтали от левого края окна браузера до контента;
- ✓ `link` — устанавливает цвет ссылок на веб-странице.
- ✓ `rightmargin` — отступ от правого края окна браузера до контента;
- ✓ `scroll` — управляет отображением полос прокрутки (задаются значения "`yes`" или "`no`");
- ✓ `text` — устанавливает цвет текста в документе;
- ✓ `topmargin` — задает отступ от верхнего края окна браузера до контента;
- ✓ `vlink` — устанавливает цвет посещенных ссылок.

Цвета атрибутов задаются в виде значений цветовой модели RGB (см. п. 2.4.2), а отступы — в единицах измерения, рассматриваемых в п. 2.4.1.

🖫 **Элемент `title`** необходим для размещения заголовка веб-страницы. Строка текста, расположенная внутри этого элемента, отображается не в документе, а в заголовке окна (или вкладки) браузера.

Допускается использовать только одну связку тегов `<title>…</title>` на веб-документ и размещать его в контейнере `head`.

Содержимое элемента часто используется при организации поиска и фактически является текстом ссылки в таблице поисковой выдачи, по которой пользователь переходит на выбранную веб-страницу.

Поэтому эта строка должна точно отражать суть документа и не быть слишком длинной. Например,

```
<title> Резюме –
        Иваненко Иван Иванович –
        Оптимизация аппаратных затрат в логических
        схемах устройств управления
</title>
```

🖫 **Элемент `style`** применяется для определения стилей элементов веб-страницы (см. п. 2.2.3).

Теги `<style>…</style>` необходимо использовать внутри контейнера `head`. Можно задавать более чем один элемент `style`.

Содержимое элемента записывается в синтаксисе каскадных таблиц CSS (см. следующую главу).

Использует атрибут `type`, который сообщает браузеру, какой синтаксис использовать, чтобы правильно интерпретировать стили (в версии HTML 5 атрибут `type` можно не устанавливать). Например,

```
< style type="text/css">
    a  {  text-decoration: none;  }
    h1 {
          color: red;
          font-size: 18pt;
    }
</style>
```

🖫 **Элемент `link`** описывает взаимосвязь документа с другими документами на сайте (например, файлы со стилями или со шрифтами), указывая его место в иерархической структуре сайта.

Элемент не имеет конечного тега, поскольку не имеет контента В заголовке может содержаться несколько элементов `link`. Тег `<link>` размещается всегда внутри контейнера `head`.

Например, с помощью элемента `link` можно подключить внешний файл, содержащий таблицы стилей CSS:

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

🖫 **Элемент `meta`** содержит служебную информацию, которая не отражается при просмотре веб-страницы. Внутри него нет текста, который бы отображался на странице, поэтому нет и закрывающего тега.

Каждый элемент `meta` содержит два основных атрибута:

✓ `name` или `http-equiv` — категория параметра;

✓ `content` — содержание параметра.

Атрибут `name` используется для получения дополнительной информации о веб-страницах и их упорядочения. Его часто заменяют атрибутом `http-equiv`, который используется для создания дополнительных полей при выполнении запроса.

Разрешается использовать более чем один элемент `meta`, и все они размещаются в контейнере `head`.

По сути `meta` определяет **метатеги**, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Примеры использования метатегов:

– адрес электронной почты автора:

```
<meta name="Reply-to"
      content="Имя@Адрес">
```

– имя автора веб-страницы:

```
<meta name="Author"
      content="Имя автора">
```

– набор ключевых слов для поиска:

```
<meta name="Keywords"
      content="слово1, слово2, слово3, …">
```

– краткое описание содержания веб-страницы:

```
<meta name="Description"
      content="Содержание страницы">
```

– указание кодировки для веб-страницы:

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

1.3. Основы HTML

HTML-документ состоит из дерева HTML-элементов и текста. Каждый элемент обозначается в исходном документе начальным (открывающим) и конечным (закрывающим) тегом. Начальный тег показывает, где начинается элемент, конечный — где заканчивается.

Теги могут вкладываться друг в друга, например,

```
<p>
  <span>
    <em>Текст</em>
  </span>
</p>
```

При вложении следует соблюдать порядок их закрытия (*принцип «матрешки»*), например, следующая запись будет **неверной**:

```
<p>
  <span>
    <em>Текст</span>
</p>
   </em>
```

Браузер *просматривает (интерпретирует)* HTML-документ, выстраивая его структуру (DOM) и отображая ее в соответствии с инструкциями, включенными в этот файл (таблицы стилей, скрипты). Если разметка правильная, то в окне браузера будет отображена HTML-страница, содержащая HTML-элементы — заголовки, таблицы, изображения и т. д.

Процесс *интерпретации (парсинг)* начинается прежде, чем веб-страница полностью загружена в браузер. Браузеры обрабатывают HTML-документы последовательно, с самого начала, при этом обрабатывая CSS и соотнося таблицы стилей с элементами страницы.

1.3.1. Дерево HTML-документа

Дерево документа (англ. *Document Tree*) — это схема построения документа HTML, которая показывает связи между различными элементами страницы: *порядок следования и вложенность элементов*.

Эта схема помогает ориентироваться в «хаотичности» размещения тегов HTML.

Между элементами дерева документа существуют определенные «родственные» связи. Рассмотрим родственные связи элементов на следующем примере кода (дерево документов, построенное для этого кода показано на рис. 1.26):

HTML

```
<html>
<head>
   <title>Заголовок страницы</title>
</head>
<body>
   <div class="mainWrap">
      <h1>Основной заголовок</h1>
      <p>Абзац текста.</p>
      <ul>
         <li>пункт 1</li>
         <li>пункт 2</li>
      </ul>
   </div>
   <div class="sideBar">
      <h2>Второй заголовок</h2>
      <p>Текст</p>
   </div>
</body>
</html>
```

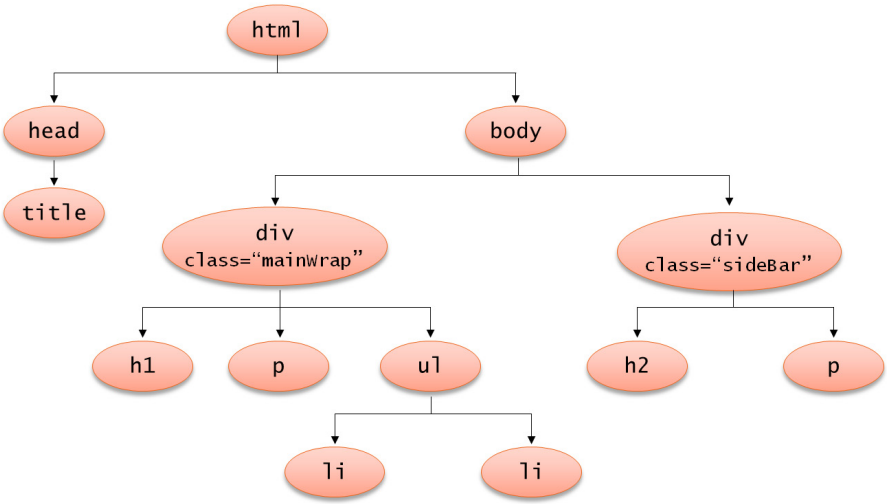


Рис. 1.26. Дерево документа HTML

Элементы, которые содержат другие, являются **предками (*ancestor*)** по отношению ко всем вложенным в него. Вложенные элементы, в свою очередь, являются его **потомками (*descendant*)**. Каждый предок может иметь неограниченное число потомков. Каждый потомок будет иметь число предков в зависимости от структуры дерева (рис. 1.27).

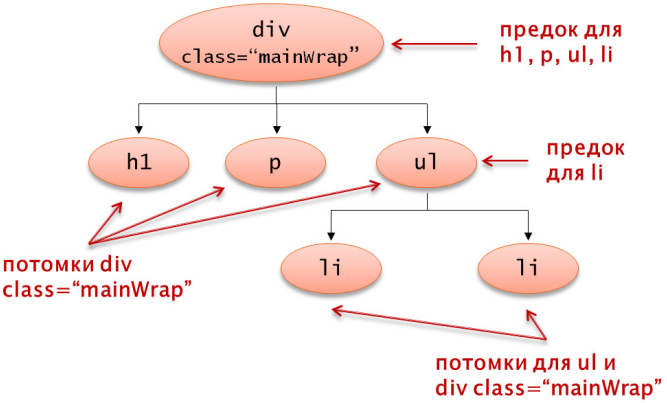


Рис. 1.27. Отношения предки-потомки в дереве документа HTML

Родитель (*parent*) — это непосредственный предок (*предок первого уровня*) элемента. И наоборот, непосредственный потомок (*потомок первого уровня*) называют **дочерним элементом (*child*)**. Отношения родительских и дочерних элементов в дереве документа показаны на рис. 1.28.

Родительский элемент еще называют **прямым предком**, а дочерний элемент — **прямым потомком**.

Сестринские элементы (*siblings*) — это группа из двух и более элементов, у которых общий родитель. Элементы не обязательно должны быть одного типа, просто у них должен быть общий родитель (рис. 1.29).

Смежные элементы (*adjacent*) — это сестринские элементы, которые расположены «по соседству» (рис. 1.30).

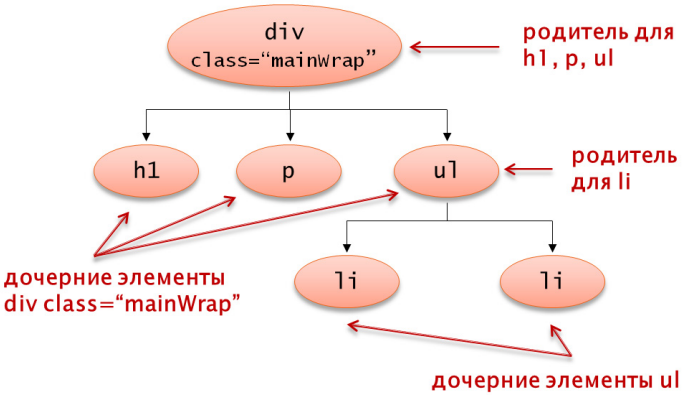


Рис. 1.28. Отношения родительских и дочерних элементов в дереве документа HTML

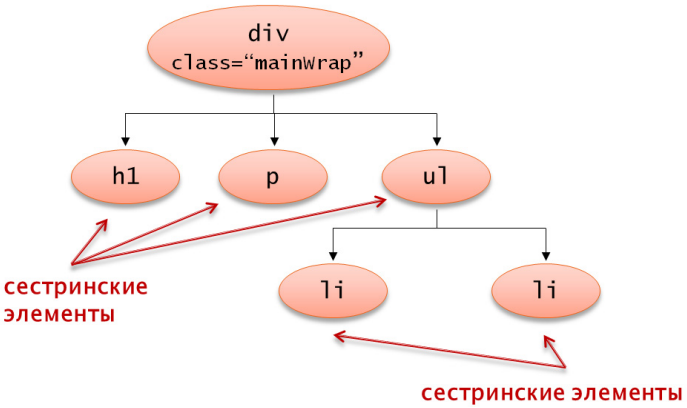


Рис. 1.29. Сестринские элементы в дереве документа HTML

Родственные связи необходимы для соблюдения **принципа наследования** свойств между элементами HTML.

Наследование — перенос правил форматирования для элементов, находящихся внутри других.

Например, дочерние элементы наследуют некоторые стилевые свойства своих родителей, внутри которых располагаются.

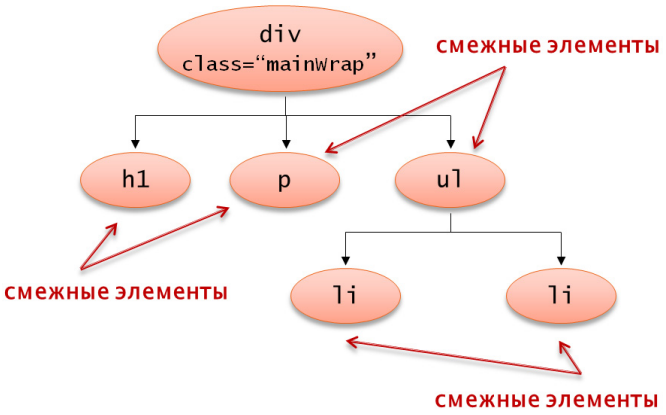


Рис. 1.30. Смежные элементы в дереве документа HTML

Рассмотрим наследование на примере (рис.1.31). Элементу body задан красный цвет отображения текста (color:red). Дочернему элементу p (в коде открывающий тег <p> занимает строку 10) цвет не задан, поэтому он наследует цвет текста у своего родителя — элемента body. Поэтому у абзаца p красный цвет текста.

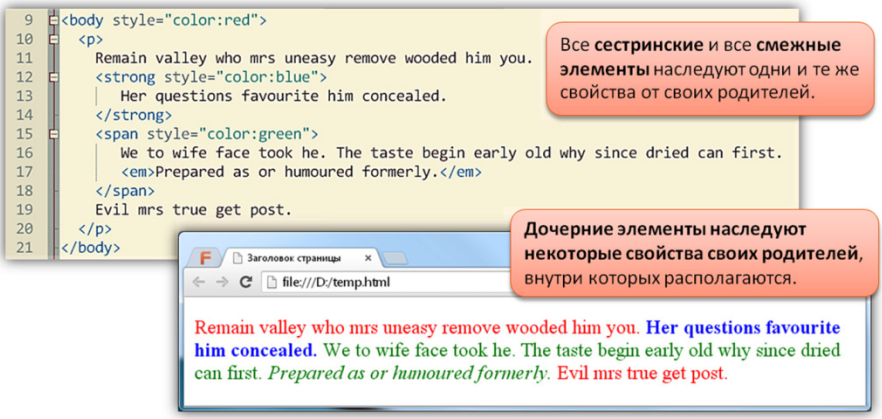


Рис. 1.31. Принципы наследования цвета текста (в браузере текст первого предложения – красный, второго – синий, третьего и четвертого – зеленый, пятого – снова красный)

В строках 12–14 встречается элемент `strong`, который является дочерним по отношению к элементу `p`. Если бы у него не было переопределения цвета, он бы наследовал красный цвет текста у своего родителя. Однако для отображения его содержимого задан синий цвет текста (`color:blue`). В строках 15–18 размещен элемент `span`, который также является дочерним по отношению к элементу `p`. Если бы у него не было переопределения цвета, он бы наследовал красный цвет текста у своего родителя. Однако для него задан зеленый цвет текста (`color:green`). Последнее предложение (строка 19) относится к абзацу и ему возвращен красный цвет текста.

1.3.2. Таблицы элементов и атрибутов

На сайте консорциума W3C (w3c.org) размещена спецификация действующей версии HTML, которая также включает полные списки атрибутов и элементов HTML 4.01.

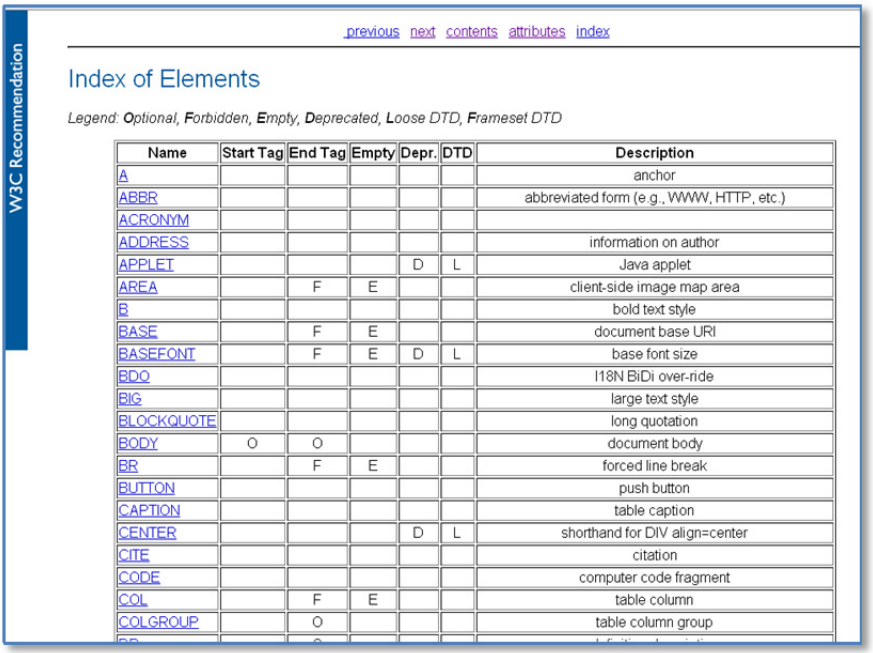
Таблица элементов размещена на странице Консорциума [24], а ее часть представлена на рис. 1.32. Каждый элемент представлен в виде ссылки на страницу с подробным описанием свойств элемента, в столбцах прописаны необходимые метки, согласно легенде, размещенной перед таблицей:

- ✓ O — не обязательно, опционально (от *Optional*);
- ✓ F — запрещено (от *Forbidden*);
- ✓ E — пустой (от *Empty*);
- ✓ D — устаревший (от *Deprecated*).

В столбце *«DTD»* могут встретиться следующие метки:

- ✓ L — переходной синтаксис HTML (*Loose DTD*);
- ✓ F — используется с применением фреймов (*Frameset DTD*).

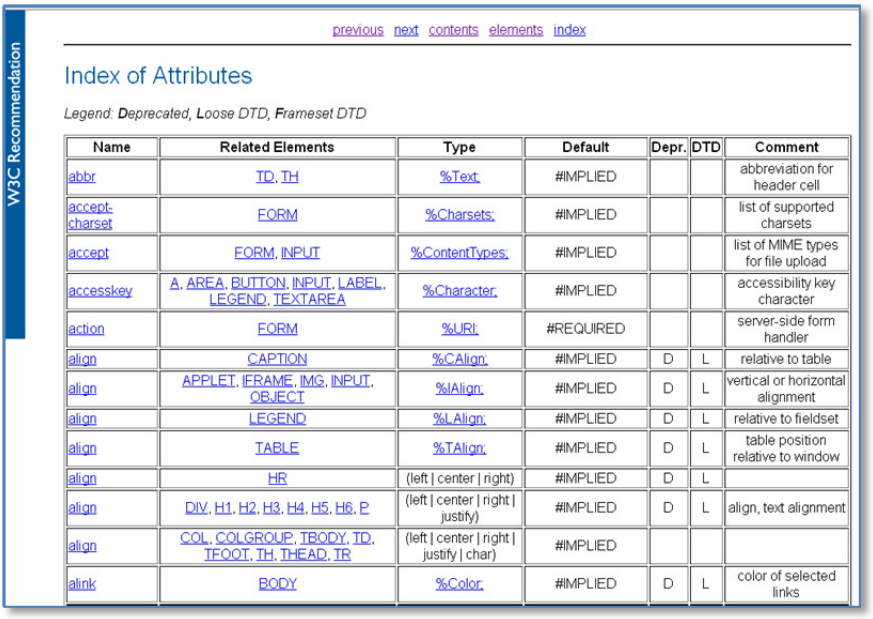
Нетрудно заметить попарную установку меток. Например, если элемент пустой (буква E в столбце *«Empty»*), то, очевидно, закрывающий тег у него запрещен (буква F в столбце *«End Tag»*). Аналогично и с устаревшими элементами: если в столбце *«Depr.»* стоит буква D, то в столбце *«DTD»* будет обязательно стоять буква L, обозначающая переходный синтаксис HTML.



Name	Start Tag	End Tag	Empty	Depr.	DTD	Description
A						anchor
ABBR						abbreviated form (e.g., WWW, HTTP, etc.)
ACRONYM						
ADDRESS						information on author
APPLET				D	L	Java applet
AREA		F	E			client-side image map area
B						bold text style
BASE		F	E			document base URI
BASEFONT		F	E	D	L	base font size
BDO						I18N BiDi over-ride
BIG						large text style
BLOCKQUOTE						long quotation
BODY	O	O				document body
BR		F	E			forced line break
BUTTON						push button
CAPTION						table caption
CENTER				D	L	shorthand for DIV align=center
CITE						citation
CODE						computer code fragment
COL		F	E			table column
COLGROUP		O				table column group

Рис. 1.32. Таблица элементов HTML действующей версии 4.01 (названия столбцов: «Name» – имя, «Start Tag» – открывающий тег, «End Tag» – закрывающий тег, «Empty» – пустой (без контента), «Depr.» – устаревший, «DTD» – версия синтаксиса, «Description» – описание

В **таблице атрибутов** (рис. 1.33), которая также размещена на странице Консорциума [25], можно выяснить назначение всех допустимых в HTML 4.01 атрибутов с указанием группы значений (в поле «*Type*» обозначены с начальным символом «%»), а также, для каких элементов они применяются (столбец «*Related Elements*»). Поле «*Default*» определяет значение по умолчанию для каждого атрибута, при этом встречаются два ключевых значения: #REQUIRED — атрибут обязателен; #IMPLIED — атрибут не обязателен. Столбцы таблицы «*Depr.*» и «*DTD*» аналогичны одноименным столбцам из таблицы элементов. В столбце «*Comment*» размещены краткие сведения по использованию каждого атрибута.



W3C Recommendation

previous next contents elements index

Index of Attributes

*Legend: **D**eprecated, **L**oose DTD, **F**rameset DTD*

Name	Related Elements	Type	Default	Depr.	DTD	Comment
abbr	TD, TH	%Text;	#IMPLIED			abbreviation for header cell
accept-charset	FORM	%Charsets;	#IMPLIED			list of supported charsets
accept	FORM, INPUT	%ContentTypes;	#IMPLIED			list of MIME types for file upload
accesskey	A, AREA, BUTTON, INPUT, LABEL, LEGEND, TEXTAREA	%Character;	#IMPLIED			accessibility key character
action	FORM	%URI;	#REQUIRED			server-side form handler
align	CAPTION	%CAlign;	#IMPLIED	D	L	relative to table
align	APPLET, IFRAME, IMG, INPUT, OBJECT	%IAlign;	#IMPLIED	D	L	vertical or horizontal alignment
align	LEGEND	%LAlign;	#IMPLIED	D	L	relative to fieldset
align	TABLE	%TAlign;	#IMPLIED	D	L	table position relative to window
align	HR	(left | center | right)	#IMPLIED	D	L	
align	DIV, H1, H2, H3, H4, H5, H6, P	(left | center | right | justify)	#IMPLIED	D	L	align, text alignment
align	COL, COLGROUP, TBODY, TD, TFOOT, TH, THEAD, TR	(left | center | right | justify | char)	#IMPLIED			
alink	BODY	%Color;	#IMPLIED	D	L	color of selected links

Рис. 1.33. Таблица атрибутов HTML 4.01

Итак, исходя из приведенных таблиц, язык HTML версии 4.01 содержит **91 элемент** и **188 атрибутов**! Действительно ли веб-разработчику необходимо досконально изучить все без исключения элементы и атрибуты? Конечно, нет! И здесь играет роль частотность применения тех или иных элементов и атрибутов.

В этом случае рекомендуется воспользоваться **принципом 80/20**, который может использоваться как базовая установка в анализе факторов эффективности какой-либо деятельности и оптимизации ее результатов:

> **Закон Парето** (*принцип Парето*, *принцип 80/20*) — эмпирическое правило, названное в честь экономиста и социолога Вильфредо Парето, в наиболее общем виде формулируется как *«20 % усилий дают 80 % результата, а остальные 80 % усилий — лишь 20 % результата»*.

Таким образом, перефразируя закон Парето, можно прийти к заключению, что **в 80% случаев достаточно около 20% из всех элементов**:

1) обязательные (html, head, body, title, link, meta);
2) основные для оформления текста (5–7 элементов);
3) дополнительные для работы с текстом (5–7 элементов);
4) важнейшие из числа прочих (около 10 элементов).

1.3.3. Адресация в HTML

При разработке сайта часто приходится прописывать пути к файлам, ссылки на документы, другие веб-страницы. В HTML применяется **абсолютная и относительная адресация**. Абсолютные адреса должны начинаться с указания протокола (обычно `http://`) и содержать имя сайта. Относительные ссылки ведут отсчет от корня сайта или текущего документа.

Адресация в HTML, в большей степени — абсолютная, формируется согласно системе **URL**, которая применяется для обозначения адресов почти всех ресурсов Интернета.

> **Унифицированный указатель ресурса** (от англ. ***U**niform **R**esource **L**ocator*, сокр. **URL**) — система унифицированных адресов электронных ресурсов, единообразный определитель местонахождения ресурса.

Локатор URL был изобретен Тимом Бернерсом-Ли в 1990 году. Изначально URL был разработан как система для максимально естественного указания на местонахождения ресурсов в сети. Локатор должен был быть легко расширяемым и использовать лишь ограниченный набор символов (к примеру, в URL никогда не применяется пробел). Синтаксис традиционной формы записи URL:

```
<протокол>://[//[<логин>[:<пароль>]@]<хост>[:<порт>]]
            [/<URL-путь>][?<параметры>][#<якорь>]
```

Поля локатора URL (заключенные в квадратные скобки поля являются не обязательными):

✓ `<схема>` — схема обращения к ресурсу (сетевой протокол);

✓ `<логин>` — имя пользователя, используемое для доступа к ресурсу;

✓ `<пароль>` — пароль указанного пользователя;

✓ `<хост>` — доменное имя в системе DNS[1] или IP-адрес[2] хоста[3];

✓ `<порт>` — порт хоста для подключения;

✓ `<URL-путь>` — уточняющая информация о месте нахождения ресурса (зависит от сетевого протокола).

✓ `<параметры>` — строка запроса с передаваемыми на сервер параметрами. Начинается с символа «?», разделителем параметров выступает знак «&»:

```
?параметр_1=значение_1&параметр_2=значение_2
```

✓ `<якорь>` — идентификатор «якоря» с предшествующим символом решетки «#». Якорем может быть указан заголовок внутри документа или атрибут `id` элемента (см. п. 2.3.3). По такой ссылке браузер откроет страницу и переместит окно к указанному элементу. Например, ссылка на страницу Консорциума W3C с якорем `#Members`:

```
https://www.w3.org/2020/Process-20200915/#Members
```

---

[1] DNS (англ. *Domain Name System* — «система доменных имен») — распределенная система для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты и/или обслуживающих узлах для сетевых протоколов.

[2] IP-адрес (от англ. *Internet Protocol Address* — «адрес Интернет-протокола») — уникальный сетевой адрес узла в компьютерной сети.

[3] Хост (от англ. *host* — «хозяин, принимающий гостей») — любое устройство, предоставляющее сервисы технологии «Клиент-Сервер» (см. п. 1.2.2) в режиме сервера.

На рис. 1.34 изображен пример URL-адреса с разделением имеющихся полей.

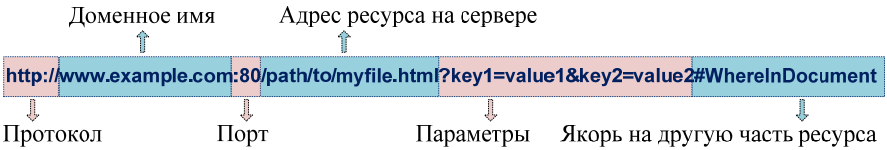


Рис. 1.34. Пример локатора URL с обозначением полей

В настоящее время URL позиционируется как часть более общей системы идентификации ресурсов URI, а сам термин URL постепенно уступает место более широкому термину **URI**.

> **Унифицированный (единообразный) идентификатор ресурса** (от англ. *Uniform Resource Identifier*, сокр. **URI**) — последовательность символов, идентифицирующая абстрактный или физический ресурс.

По сути, URI является символьной строкой, позволяющей идентифицировать какой-либо ресурс: документ, изображение, файл, службу, ящик электронной почты и т. д. Прежде всего это ресурсы Всемирной паутины.

Но все-таки, *абсолютная адресация в HTML — это ссылка, записанная с помощью системы URL*. При этом абсолютная адресация применяются в первую очередь для указания на другой сетевой ресурс и достаточно редко используются в рамках одного сайта. Еще ряд примеров абсолютной адресации:

– путь к html-файу по протоколу HTTP:

```
http://www.site.ru/page1.html
```

– путь к zip-файлу по протоколу HTTP:

```
http://www.site.ru:81/test.zip
```

– путь к исполняемому exe-файлу по протоколу FTP:

```
ftp://site.ru/pub/install.exe
```

На рис. 1.35 показаны принципы формирования *относительных адресов* на примере значения атрибута `href`. Здесь нужно учитывать некоторые особенности:

1) если файлы располагаются в одной папке, то необходимо сделать ссылку из исходного документа на ссылаемый:

```
href="target.html"
```

2) файлы размещаются в разных папках, но исходный файл располагается в корне сайта, а файл, на который необходимо сделать ссылку — в папке:

```
href="folder/target.html"
```

3) файлы размещаются в разных папках, но исходный документ хранится в одной папке, а ссылаемый в корне сайта. Тогда перед именем файла в адресе ссылки следует поставить две точки и слеш — «`../`»:

```
href="../target.html"
```

4) файлы размещаются в разных папках, но исходный документ хранится в одной папке, а ссылаемый в другой папке. Тогда перед именем файла в адресе ссылки следует поставить «`../`» (выйти на уровень выше) и далее указать папку и ссылаемый документ:

```
href="../folder/target.html"
```

5) если файл находится внутри не одной, а двух папок, то путь к нему записывается так:

```
href="folder1/folder2/target.html"
```

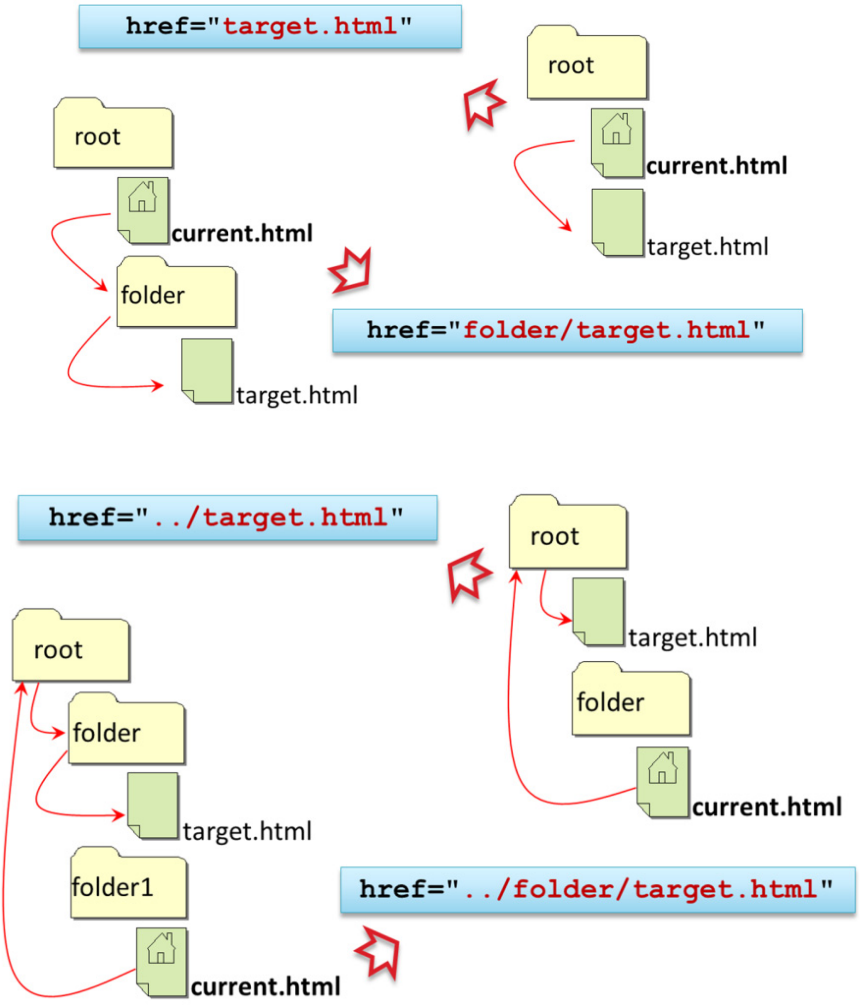


Рис. 1.35. Примеры относительной адресации в HTML

6) если исходный файл находится в двух вложенных папках, и чтобы сослаться на документ в корне сайта, требуется повторить две точки и слеш два раза:

```
href="../../target.html"
```

Иногда можно встретить путь к файлу **относительно корня сайта**. Записывается путь со слешем вначале:

```
href="/folder/target.html"
```

Например, запись

```
href="/anyfiles/"
```

означает, что ссылка ведет в папку с именем `anyfiles`, которая располагается в корне сайта, а в ней необходимо загрузить индексный файл. Однако следует учитывать, что такая форма записи *не работает на локальном компьютере, а только под управлением веб-сервера*.

1.3.4. Организация гиперссылок

Гиперссылки позволяют переходить с одной веб-страницы на другую. При этом ссылка может вести не только на файлы HTML, но и на файл любого типа, причем этот файл может размещаться на другом сайте. Главное, чтобы к веб-объекту, на который делается ссылка, был доступ — *если путь к веб-объекту доступен через адресную строку браузера, и веб-объект при этом в наличии (т. е. может быть открыт), то на него можно сделать гиперссылку*.

Для создания гиперссылки необходимо:

– сообщить браузеру, что является ссылкой (текст, изображение и пр.);

– указать URL-адрес документа, на который следует сделать ссылку.

Организация этих действий осуществляется с помощью тегов `<a>…</a>`.

🖫 **Элемент `a`** (от англ. ***a**nchor*) является одним из важных элементов HTML и предназначен для создания гиперссылок. Общий синтаксис:

```
<a href="URL-адрес">текст ссылки</a>
```

Для создания гиперссылки необходимо сообщить браузеру, что является ссылкой, а также указать URL документа, на который происходит переход. Адрес ссылки может быть *абсолютным* и *относительным* (см. п. 1.3.3).

Основные атрибуты элемента `a`:

✓ `href` — задает адрес документа, на который следует перейти. Поскольку в качестве адреса ссылки может использоваться документ любого типа, то результат перехода по ссылке зависит от конечного файла. Так, архивы (например, файлы `.zip` или `.rar`) будут сохраняться на локальный диск компьютера-клиента. По умолчанию новый документ загружается в текущее окно браузера, однако это свойство можно изменить с помощью атрибута `target`.

✓ `target` — указывает, в каком окне (вкладке) браузера осуществлять переход по ссылке (в XHTML применение этого атрибута запрещено). Допустимые значения атрибута:

– "`_blank`" — загружает страницу в новую вкладку браузера;
– "`_self`" — загружает страницу в текущее окно (значение по умолчанию);
– "`_parent`" — загружает страницу во фрейм-родитель (если фреймов нет, то это значение работает как `_self`);
– "`_top`" — отменяет все фреймы и загружает страницу в полном окне браузера, если фреймов нет, то это значение работает как `_self`.

Использование этого атрибута осуждается спецификацией HTML. Для обеспечения валидного кода (см. п. 1.3.6) необходимо использование переходного синтаксиса HTML (см. п. 1.2.4, табл. 1.1).

✓ `name` — используется для определения якоря внутри страницы.

В зависимости от присутствия атрибутов `name` или `href` элемент `a` устанавливает **гиперссылку** (строка 1) или **якорь** (строка 2):

HTML

```
1  <a href="URL"> ... </a>
2  <a name="идентификатор"> ... </a>
```

Пример гиперссылки по URL-адресу портала магистров Донецкого национального технического университета. Здесь используется универсальный атрибут `title` (см. п. 1.3.5), который добавляет всплывающую подсказку к тексту ссылки:

HTML

```
<a href="http://masters.donntu.org"
   target= "_blank"
   title="Перейти на сайт портала магистров
   Донецкого национального технического
   университета">Портал магистров ДонНТУ</a>
```

Рассмотрим основные принципы организации **якорных гиперссылок**.

Якорь (англ. *anchor*) — закладка внутри страницы, которую можно указать в качестве цели ссылки. При использовании ссылки, которая указывает на якорь, происходит переход к закладке внутри веб-страницы.

Вначале следует задать в соответствующем месте закладку и установить ее имя при помощи атрибута `name` тега `<a>`. Имя ссылки на закладку начинается символом #, после чего идет соответствующее тематике название закладки. Можно также создать ссылку на закладку, находящуюся в другой веб-странице

текущего или любого другого сайта. Для этого в адресе ссылки нужно указать ее URL-адрес, а в конце добавить символ решетки # и имя закладки.

Между тегами и текст писать не обязательно, так как требуется лишь указать местоположение перехода по ссылке (рис. 1.36).

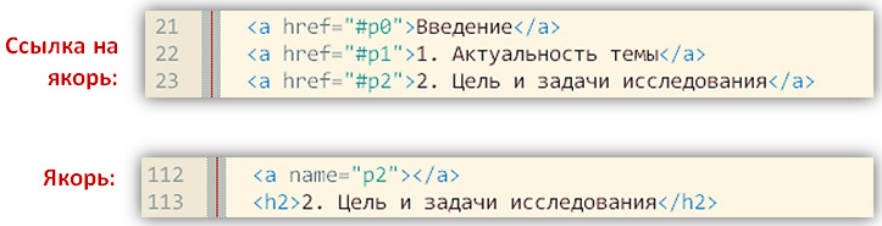


Рис. 1.36. Пример организации якорных гиперссылок

Создание **ссылки на адрес электронной почты** делается аналогично ссылке на веб-страницу. Только вместо адреса URL указывается конструкция `mailto:адрес_электронной_почты`. Например,

HTML

```
<a href="mailto:ivanitsa-serg@rambler.ru">
   ivanitsa-serg@rambler.ru</a>
```

Такая ссылка визуально не отличается от ссылки на веб-страницу, но при нажатии на нее *запускается почтовая программа, установленная на компьютере клиента по умолчанию*. Поэтому в названии ссылки следует указывать, что она имеет отношение к электронной почте (или саму электронную почту, как в примере выше), чтобы пользователи понимали к чему приведет нажатие на нее.

Можно также автоматически добавить тему будущего письма, присоединив к адресу электронной почты через символ вопроса «?» — параметр `subject="тема сообщения"`. При запуске почтовой программы поле *Тема (Subject)* будет заполнено автоматически. Например,

HTML

```
<a href="mailto:ivanitsa-serg@rambler.ru?Утверждение
   темы индивидуального раздела персонального сайта">
   ivanitsa-serg@rambler.ru</a>
```

Любая гиперссылка на веб-странице может находиться в одном трех состояний.

Непосещенная ссылка. Такое состояние характеризуется для ссылок, которые еще не открывали. *По умолчанию непосещенные текстовые ссылки изображаются синим цветом и с подчеркиванием*.

Активная ссылка. Ссылка помечается как активная в момент ее открытия. Поскольку время между нажатием на ссылку и началом загрузки нового документа достаточно мало, подобное состояние ссылки весьма кратковременно. Активной ссылка становится также, при ее выделении с помощью клавиатуры. *Цвет такой ссылки по умолчанию красный*.

Посещенная ссылка. Как только пользователь открывает документ, на который указывает ссылка, она помечается как посещенная и *меняет свой цвет на фиолетовый, установленный по умолчанию*.

Форматирование состояния ссылок (например, изменение цвета, убирание или изменение вида подчеркивания и пр.) возможно с помощью стилевых свойств CSS при использовании соответствующих псевдоклассов (см. п. 2.3.9).

1.3.5. Универсальные атрибуты

Универсальные атрибуты применяются практически ко всем тегам, поэтому выделены в отдельную группу. Перечислим основные универсальные атрибуты (сюда не вошли специфические универсальные атрибуты, используемые только в HTML 5 и не поддерживаемые другими версиями языка гипертекстовой разметки):

✓ `style` — применяется для определения стиля элемента с помощью правил CSS (см. п. 2.2.3);

✓ `class` — определяет имя класса, которое позволяет связать элемент HTML со стилевым оформлением CSS (см. п. 2.3.2);

✓ `id` — указывает имя стилевого идентификатора (см. п. 2.3.3);

✓ `dir` — задает направление и отображение текста — слева направо (значение "`ltr`" — значение по умолчанию) или справа налево (значение "`rtl`"). Браузеры обычно самостоятельно различают направление текста, если он задан в кодировке Юникод, но с помощью атрибута `dir` можно указать, в каком направлении отображать текст. Для арабских и еврейских символов приоритетным является направление, заложенное в Юникод, поэтому на них значение атрибута `dir` не воздействует;

✓ `hidden` — скрывает содержимое элемента от просмотра. Это **логический атрибут**, т. е. не принимающий конкретного значения, однако наличие только имени атрибута в теге элемента говорит об истине «yes» его назначения, а отсутствие — о лжи «no», когда назначение атрибута не выполняется. В качестве значения (актуально для всех логических атрибутов) можно также указать само имя `hidden` (`hidden="hidden"`) или оставить атрибут пустым (`hidden=""` или `hidden`):

HTML

```
<div hidden>
   <p>Так скрываются элементы, включая этот параграф,
      поскольку скрыт его родитель</p>
</div>
<a hidden="hidden" href="main.my">Скрытая ссылка</a>
```

✓ `lang` — устанавливает *код языка*[1] для правильного отображения некоторых национальных символов браузером. Текст документа может быть набран как на одном языке, так и содержать вставки на других языках, которые могут различаться по своим

[1] Двухбуквенные коды языков приняты стандартом ISO 639. Языковые коды, установленные в разделах ISO 639, используется для библиографических целей, как ключевые элементы языковых данных в компьютерных и интернет-средах. На базе ISO 639 основывается стандарт RFC 3066, описывающий использование кодов языков в Интернете.

правилам оформления текста. Например, *для русского, немецкого и английского языка характерны разные символы кавычек*, в которые берется цитата. Чтобы указать язык, на котором написан текст внутри текущего элемента и применяется `lang`;

✓ `tabindex` — устанавливает порядок получения фокуса при переходе между элементами с помощью клавиши *Tab*:

HTML

```
<p>Нажмите кнопку Tab для перехода между элементами</p>
<p><button tabindex="6">Шестой</button></p>
<p><button tabindex="7">Седьмой</button></p>
<p><button tabindex="5">Пятый</button></p>
<p><button tabindex="1">Первый</button></p>
<p><button tabindex="3">Третий</button></p>
<p><button tabindex="2">Второй</button></p>
<p><button tabindex="4">Четвертый</button></p>
```

✓ `title` — описывает содержимое элемента в виде всплывающей подсказки, которая появляется при наведении курсора на элемент. Стили отображения всплывающей подсказки зависят от браузера, настроек операционной системы и не могут быть изменены с помощью HTML-кода или CSS-стилей.

HTML

```
<h1 title="Это мой заготовок!">Интернет-технологии</h1>
```

1.3.6. Комментарии в HTML

Некоторый текст в разметке HTML можно скрыть от показа в браузере, сделав его **комментарием**. Помеченный как комментарий (закомментированный) фрагмент html-файла поступит с остальным кодом в браузер, однако не будет участвовать в построении дерева документа. Поэтому этот текст пользователь не увидит, но, посмотрев исходный код веб-страницы, можно обнаружить скрытые заметки.

Комментарии нужны для внесения в код своих записей, не влияющих на вид страницы. Начинаются они символьной груп-

пой <!-- и заканчиваются группой из трех символов -->. Все, что находится между этими группами символов (условно их также называют тегами), отображаться на веб-странице не будет:

```
<!-- текст комментариев -->
```

Разрешается внутрь комментария добавлять другие теги, однако **недопустимы вложенные комментарии** (когда один комментарий расположен внутри другого). Также **не допускается** в тексте комментария **использовать двойной дефис** «--». Двойной дефис внутри комментария воспринимается как часть элемента комментария и приводит к ошибке при *валидации*[1] HTML. Примеры организации комментариев:

HTML

```
<!-- это однострочный комментарий -->

<!-- это комментарий
                  но уже многострочный  -->

<!-- это комментарий --
     содержит двойной дефис  ---->

<!-- в этом комментарии содержится код,
     который
     временно закомментирован
  <div>
     <a href="http://scholar.google.com/">
     Ссылка на Google Scholar</a>
     <p> Необходимо каждому ученому иметь аккаунт в
         сервисе Google Scholar</p>
  </div>   -->
```

[1] Валидация — это проверка веб-документа на соответствие веб-стандартам по спецификации HTML, разработанной Консорциумом W3C (w3c.org) при поддержке разработчиков браузеров. Соответственно, валидным является такой веб-документ, который прошел подобную процедуру и не имеет замечаний по коду. По адресу http://validator.w3.org располагается самый распространенный инструмент для проверки отдельных веб-страниц на валидность.

HTML

```
<!-- а так можно помечать закрывающие теги, чтобы
     не запутаться в иерархии элементов: -->
<div id="topblock">
   <div id="langbox">
      <div id="donntu">
          <div id="header">
              ...
          </div>  <!-- header -->
          ...
      </div>  <!-- donntu -->
      ...
   </div>  <!-- langbox -->
   ...
</div>  <!-- topblock -->
```

1.4. Элементы HTML

Особое место среди всех элементов HTML занимают элементы для оформления текстового содержимого веб-страниц. Действительно, не смотря на насыщенность современных сайтов графической информацией, страница сайта в среднем на 70–80 % состоит из текстового содержимого.

На рис. 1.37 представлен набор элементов для работы с текстом сайтов. Это оптимальный набор как основных (использующихся чаще всего) так и дополнительных (использующихся реже) элементов, для удовлетворения практически всех потребностей веб-разработчика для разметки текстовой части веб-страницы.

1.4.1. Основные элементы для оформления текстов

Наиболее часто в html-разметке, среди элементов, отвечающих за текстовое оформление, можно встретить два элемента: `p` (текстовый параграф) и `br` (принудительный перенос строки).

Рис. 1.37. Элементы HTML для работы с текстовым содержимым веб-страницы

🖫 **Элемент** `p` (от англ. ***paragraph***) — **создает новый параграф** (текстовый абзац). Каждый абзац текста должен быть заключен в свою пару тегов `<p>…</p>`. Однако, согласно таблице атрибутов (см. п. 1.3.2, рис. 1.32) для элемента `p` *закрывающий тег не обязателен*. Это значит, что браузер всегда определит, где заканчивается текущий абзац и поставит закрывающий тег автоматически.

У элемента `p` в качестве основного атрибута выступает параметр `align`, который задает выравнивание относительно одной из сторон документа, и принимате одно из константных значений: `left` (выравнивание слева), `right` (выравлнивание справа), `justify` (выравнивание по ширине) или `center` (выравнивание по центру). Наиболее распространенный вариант — выравнивание по левому краю (`left` — значение по умолчанию), когда слева текст сдвигается до края, а правый остается неровным. Выравнивание по правому краю и по центру в основном используется в заголовках и подзаголовках. Следует иметь в виду, что при использовании выравнивания по ширине в тексте между

словами могут появиться большие интервалы, что не очень красиво.

Содержимое элемента p всегда начинается с новой строки в окне браузера, а абзацы текста, идущие друг за другом, разделяются между собой небольшим отступом (отбивкой):

HTML

```
<body>
   <p>Абзац 1</p>
   <p align="left">Абзац 2</p>
   <p align="center">Абзац 3
   <p align="right">Абзац 4<p>Абзац 5
</body>
```

Результат приведенного кода показан на рис. 1.38.

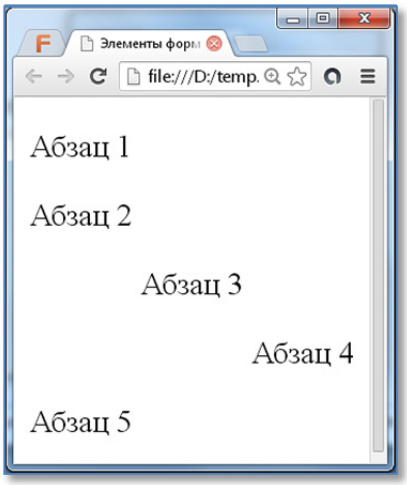


Рис. 1.38. Использование элемента p

🖫 **Элемент br** (от англ. ***br****eak*) представляет собой одиночный (непарный) тег
, который **устанавливает перевод строки** в том месте, где этот тег находится.

При работе с текстом средствами HTML, необходимо учитывать, что

1) любое количество идущих подряд пробельных символов «схлопывается» до одного пробела при отображении текста в браузере;

2) символы переноса строки и символы табуляции также заменяются одним пробельным символом.

Таким образом, если при подготовке разметки HTML в редакторе кода (например, в текстовом редакторе) были расставлены переносы, табуляция и подряд идущие пробелы, то в браузере эта разметка будет проигнорирована. В месте, где необходимо принудительно перенести строку, вставляется элемент `br` (рис. 1.39).

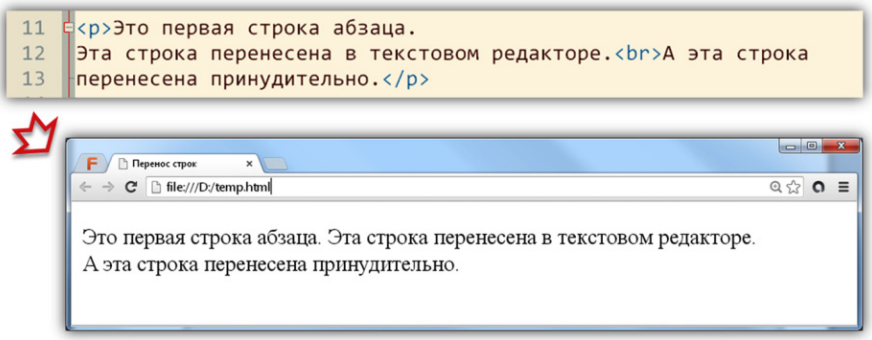


Рис. 1.39. Применение элемента br для принудительного переноса строки

В отличие от элемента абзаца `p`, использование `br` не добавляет пустой отступ (отбивку) перед строкой (рис. 1.40).

***

На рис. 1.41 показана следующая группа часто используемых элементов. При этом две пары элементов — `b` и `strong`, также как `i` и `em`, — имеют одно и то же назначение, однако являются не совсем эквивалентными и заменяемыми.

Рассмотрим пару элементов `b` и `strong`, которые устанавливают *жирное начертание текста*. Элемент `b` является элементом физической разметки, а `strong` — элементом логической разметки и определяет важность помеченного текста.

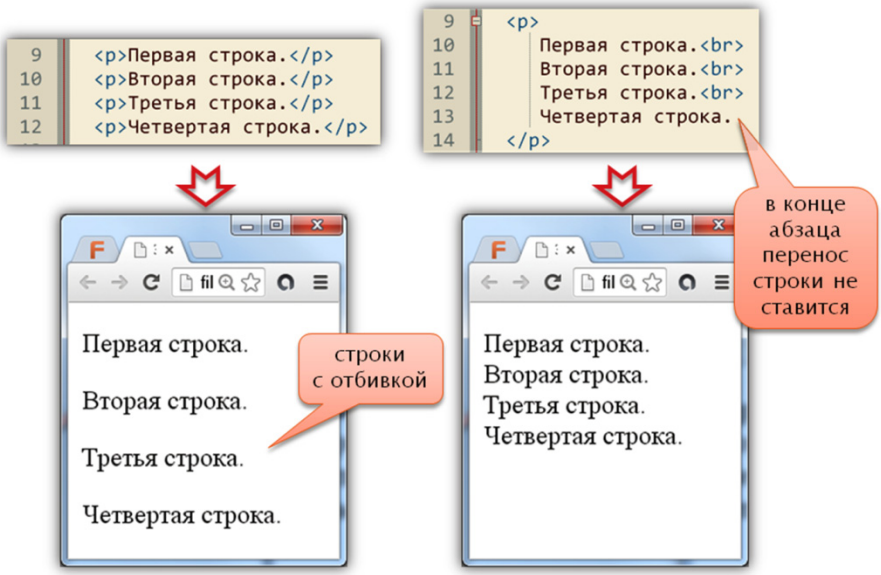


Рис. 1.40. Наличие и отсутствие отступов при использовании элементов абзаца p и переноса строки br

	Тег	Описание	Код HTML	Отображение
	pre	Отображение текста «как есть», включая все пробелы	<pre>Текст</pre>	Текст
1	b	Жирное начертание текста	Текст	**Текст**
1	i	Курсивное начертание текста	Текст	*Текст*
	sup	Верхний индекс	a² + b³	$a^2 + b^3$
	sub	Нижний индекс	SO₄	SO_4
2	strong	Жирное начертание текста	Текст	**Текст**
2	em	Курсивное начертание текста	Текст	*Текст*

Рис. 1.41. Основные элементы форматирования текста (1 – элементы физического форматирования; 2 – элементы логического форматирования)

Такое разделение тегов на логическое и физическое форматирование изначально предназначалось, чтобы сделать HTML универсальным, в том числе не зависящим от устройства вывода информации. Теоретически, если воспользоваться, например, речевым браузером, то текст, оформленный с помощью тегов `<b>…</b>` и `<strong>…</strong>`, будет отмечен по-разному. Однако получилось так, что в популярных браузерах результат использования этих тегов равнозначен [26].

Любые теги форматирования текста можно использовать совместно друг с другом. Чтобы сделать текст одновременно *жирным* (***bold***) и *курсивным* (***italic***) используется сочетание тегов `<b>…</b>` и `<i>…</i>` (их порядок в данном случае не важен). Это же относится к тегам `<strong>…</strong>` и `<em>…</em>` (рис. 1.42).

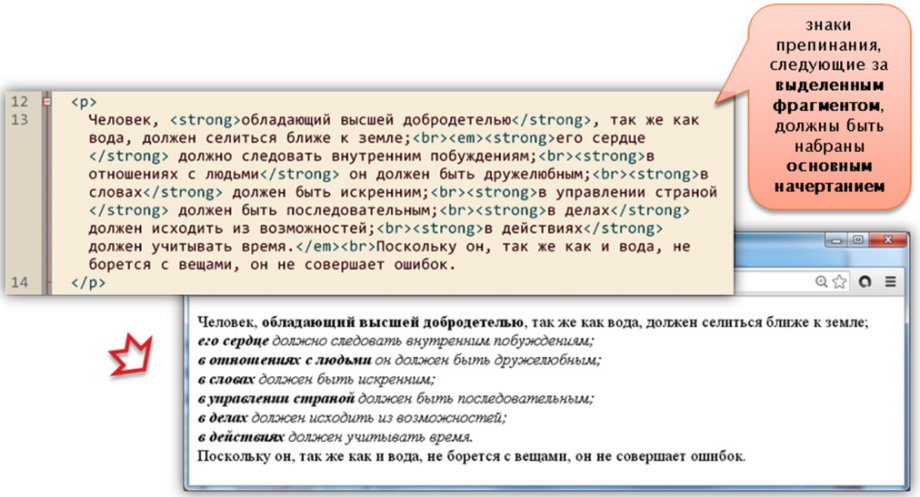


Рис. 1.42. Формирование жирного и курсивного начертаний текста

Как было отмечено ранее, в коде HTML любое количество подряд идущих пробельных символов в веб-странице показывается как один. Такова особенность отображения текста всеми браузерами. Элемент `pre` позволяет обойти эту особенность и отображать текст как требуется разработчику.

🖫 **Элемент `pre`** определяет блок предварительно форматированного текста, который обычно отображается моноширинным[1] шрифтом и со всеми пробелами между словами. Внутри элемента `pre` не допускается применять элементы `img` (*вставка изображения*), `sub` (*нижний индекс*) и `sup` (*верхний индекс*).

В следующем примере показано использование конструкции `<pre>…</pre>`, а на рис. 1.43 — результат выполнения этого примера в браузере Chrome.

HTML

```
<p>
  <pre>
     Билет –
            щелк.
                Щека –
                      чмок.
     Свисток –
              и рванулись туда мы,
     куда,
         как сельди,
                   в сети чулок
     плывут
           кругосветные дамы.
  </pre>
</p>
```

🖫 **Элементы `<h1>…</h1>`, …, `<h6>…</h6>` — шесть элементов-заголовков разного уровня**, которые показывают относительную важность текста, расположенного после заголовка.

Текст между элементами `h1`, ..., `h6` всегда начинается с новой строки, а после него другие элементы отображаются на следующей строке. Браузеры отображают тексты заголовков в жирном начертании. Кроме того, перед заголовком и после него добавляется пустое пространство.

[1] Моноширинный, или непропорциональный шрифт — это шрифт, в котором все знаки имеют одинаковую ширину. Этим он отличается от пропорционального шрифта, в котором буквы отличаются по ширине друг от друга.

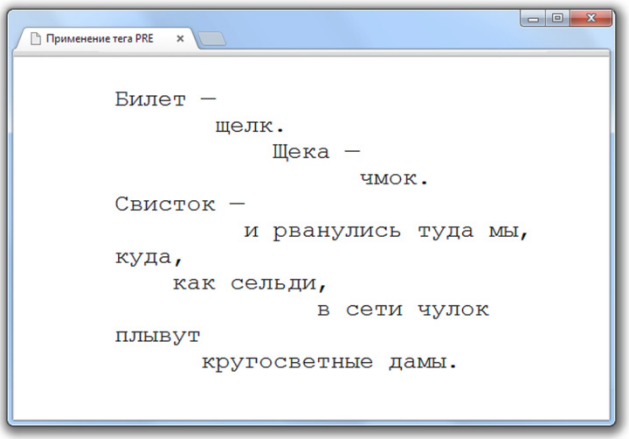


Рис. 1.43. Отображение форматированного текста с использованием элемента pre

Элемент h1 представляет собой наиболее важный заголовок первого уровня, а элемент h6 — заголовок шестого уровня и является наименее значительным. Уровни заголовков соответствуют книжной иерархии: h1 — это заголовок целой книги; h2 — заголовок части; h3 — главы, и т. д.

Хорошим тоном верстки считается использование одного заголовка первого уровня (h1) на веб-странице — это будет заголовок целого документа. В веб-документе не должно быть заголовков третьего уровня (h3), если до этого не был использован заголовок второго уровня (h2).

Заголовки мельче четвертого уровня (h5 и h6) лучше использовать только в случае крайней необходимости (их размер уже меньше размера основного текста, заданного по умолчанию).

HTML

```
<h1 align=“center”>Заголовок первого уровня (h1)</h1>
<h2 align=“center”>Заголовок второго уровня (h2)</h2>
<h3 align=“center”>Заголовок третьего уровня (h3)</h3>
<h4 align=“center”>Заголовок четвертого уровня (h4)</h4>
<h5 align=“center”>Заголовок пятого уровня (h5)</h5>
<h6 align=“center”>Заголовок шестого уровня (h6)</h6>
```

На рис. 1.44 показано выполнение выше приведенного html-кода — подряд идущие заголовки от первого до шестого уровня. Основной атрибут, применяемый к тегам заголовков, — `align`, который определяет горизонтальное выравнивание заголовков.

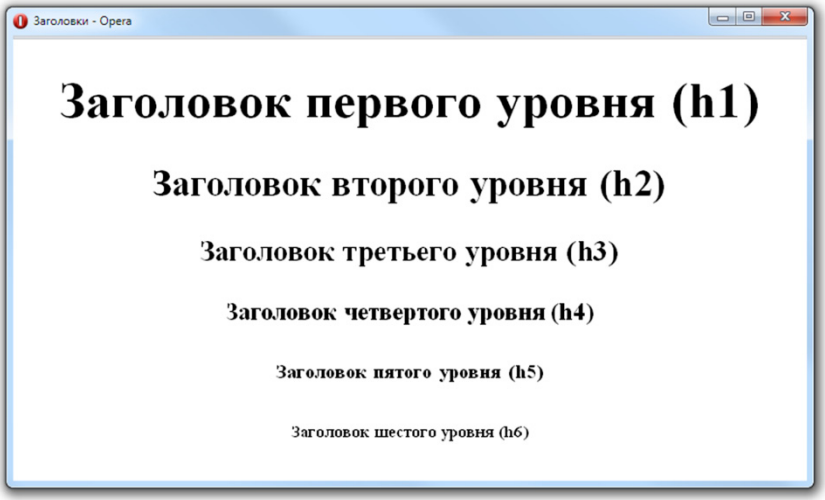


Рис. 1.44. Заголовки от первого до шестого уровня с выравниванием по центру

Заголовки очень «любят» поисковые системы, следовательно, заголовки повышают ценность текста на веб-странице, который располагается внутри элементов `h1`…`h6`. По этой причине лучше всегда использовать эти элементы, несмотря на то, что с помощью стилей CSS любой текст можно сделать заголовком.

1.4.2. Дополнительные элементы для оформления текстов

Рассмотренные выше элементы `em` и `strong` (равно как и элементы `i` и `b`) выполняют функцию **текстовых выделений**. К текстовым выделениям также относятся элементы, которые используются реже, но являются не менее актуальными:

Элемент	Значение
abbr	аббревиатура
address	информация об авторе и документе
blockquote	выделение цитат
code	фрагмент исходного кода
dfn	определение термина
kbd	текст, введенный на клавиатуре
samp	выходные данные
var	имя переменной или параметра

Например, элемент abbr рекомендуется использовать вместе с универсальными атрибутами lang и title (см. п. 1.3.5), сообщающими о языке и расшифровке аббревиатуры:

HTML

```
<abbr lang="ru" title="Донецкий национальный технический
      университет">ДОННТУ</abbr>

<abbr lang="en" title="World Wide Web">WWW</abbr>
```

Элемент abbr можно также использовать для сокращений:

HTML

```
<abbr lang="ru" title="Кандидат технических
      наук">к. т. н.</abbr>

<abbr lang="ru" title="Доктор экономических
      наук">д. э. н.</abbr>

<abbr lang="la" title="Philosophiæ Doctor">PhD</abbr>

<abbr lang="en" title=" Doctor of Science">Sc.D</abbr>
```

Действие остальных дополнительных элементов наглядно продемонстрировано на следующем примере (результат отображения данного кода в браузере показан на рис. 1.45):

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Элементы текстовых выделений</title>
  </head>
  <body>
    <p><abbr lang="ru" title="Донецкий национальный
       технический университет">ДонНТУ</abbr></p>

    <p><dfn>Терминал</dfn> – конечная часть некой
       системы, которая обеспечивает связь системы
       с внешней средой.</p>

    <blockquote cite="http://habrahabr.ru/post/141922/">
      <p>Самая маленькая группа людей, перед которыми
         мне приходилось выступать – три человека
         (переговоры один на один не считаю), самая
         большая – человек 800 в зале конференции
         Sun Tech Days в Хайдерабаде, в Индии.</p>
    </blockquote>

    <p>Элемент <code>abbr</code> рекомендуется
       Использовать вместе с атрибутами
       <code>lang</code> и <code>title</code>.</p>

    <p>В поле <samp>Имя пользователя</samp> введите
       <kbd>Administrator</kbd>.</p>
    <p>Введем переменные <var>X</var> и
       <var>Y</var>.</p>

   <blockquote>
     <p>И долго буду тем любезен я народу,<br>Что
        чувства добрые я лирой пробуждал,<br>Что в
        мой жестокий век восславил я свободу<br>
        И милость к падшим призывал.
        <address>Александр Пушкин</address></p>
    </blockquote>
  </body>
</html>
```

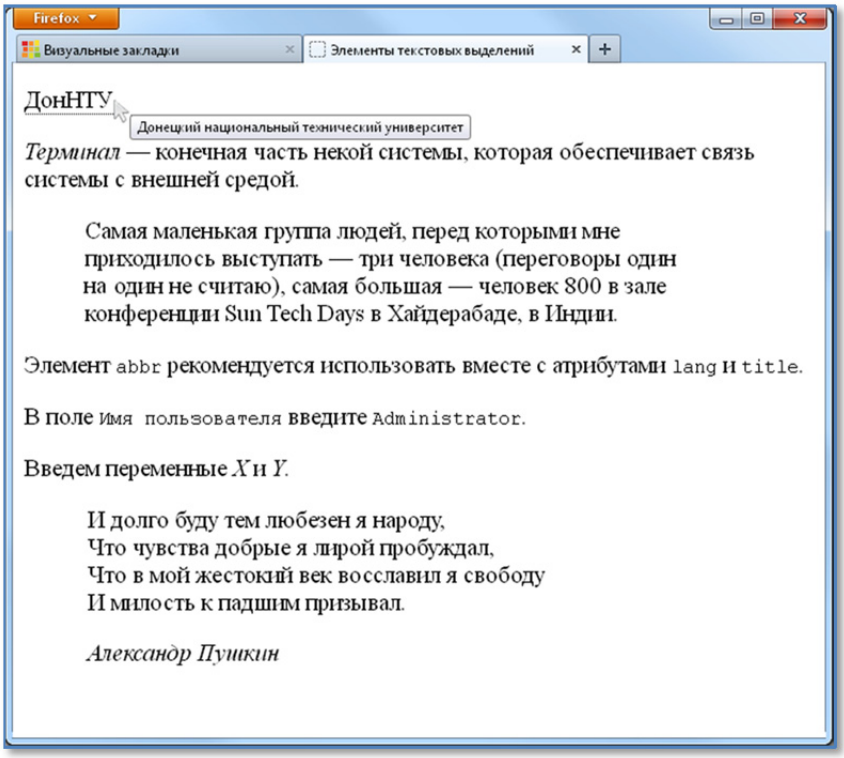


Рис. 1.45. Элементы текстовых выделений

1.4.3. Графические элементы

Под графическими элементами понимаются такие элементы HTML, которые размещают на отображенной в браузере веб-странице какой-либо графический объект. В данном контексте рассмотрению подлежат два элемента — `img` и `hr`. Особенностью этих графических элементов является естественное отсутствие текстового содержимого, и, следовательно, закрывающего тега.

Элемент `img` (от ***image***) добавляет изображение в документ HTML из графических форматов GIF, JPEG или PNG [16, 27]:

```
<img src="URL-адрес" [атрибуты]>
```

Элемент `img` определен одиночным тегом `<img>` с возможными атрибутами (рис. 1.46):

✓ `src` — путь (*абсолютный* или *относительный*) к файлу изображения, включая имя и расширение файла — без этого атрибута элемент `img` не имеет смысла;

✓ `height` и `width` — высота и ширина изображения, задаваемая доступными единицами изменения (см. п. 2.4.1);

✓ `align` — выравнивает изображение к одной из сторон документа с допустимыми значениями: `left` (*слева*), `right` (*справа*), `center` (*в центре по горизонтали*), `bottom` (*снизу*), `top` (*сверху*), `middle` (*в середине по вертикали*);

✓ `border` — устанавливает толщину рамки вокруг изображения в доступных единицах изменения;

✓ `vspace` — устанавливает поля сверху и снизу;

✓ `hspace` — устанавливает поля с боков;

✓ `name` — задает имя изображения для доступа из JavaScript-сценариев;

✓ `alt` — определяет альтернативный текст, отображаемый браузером на месте изображения, если браузер не может отобразить графический файл (файл по каким-либо причинам (удален или поврежден на сервере) не был загружен на компьютер клиента, либо в браузере установлен режим отображения страницы без графики).

Если необходимо, то рисунок можно сделать ссылкой на другой файл, поместив тег `<img>` в контейнер `<a>…</a>`. При этом вокруг изображения отображается рамка, которую можно убрать, добавив атрибут `border="0"` в тег `<img>`. В примере ниже показан вариант использования одного изображения (файл `photo.jpg`) в качестве ссылки, кликнув на которое загрузится другое изображение (файл `ivanenko_big.jpg`):

HTML

```
<a href="ivanenko_big.jpg">
    <img src="photo.jpg" border="0" width="180px"
         height="240px" title="Магистр ДонНТУ "
         alt= "Магистр ДонНТУ Иваненко Иван Иванович">
</a>
```

```
15
16   <img src="cat.jpg" name="cat" width="691" height="473"
17        border="2" hspace="50" vspace="50"
18        title="Funny picture" alt="Mad football cat">
19
```

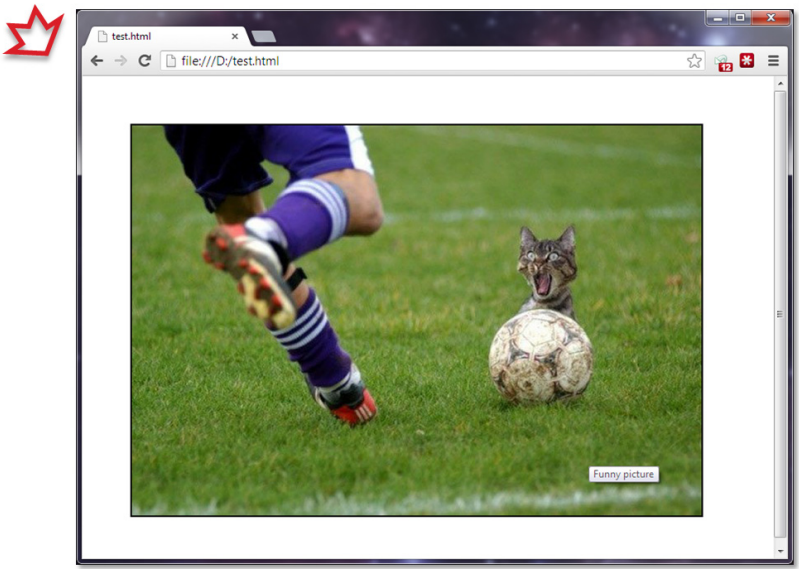


Рис. 1.46. Пример использования элемента img

Элемент `hr` (от ***h**orizontal **r**uler*) рисует горизонтальную линию, которая по своему виду зависит от используемых параметров. Линия всегда начинается с новой строки, после нее все элементы отображаются на следующей строке.

Элемент `hr` определен одиночным тегом `<hr>` со следующими необязательными атрибутами (рис. 1.47):

- ✓ `size` — устанавливает толщину линии в доступных единицах измерения (см. п. 2.4.1);
- ✓ `width` — устанавливает ширину линии в доступных единицах измерения;
- ✓ `noshade` — логический атрибут (см. п. 1.3.5), который создает линию без тени;
- ✓ `color` — задает цвет линии.

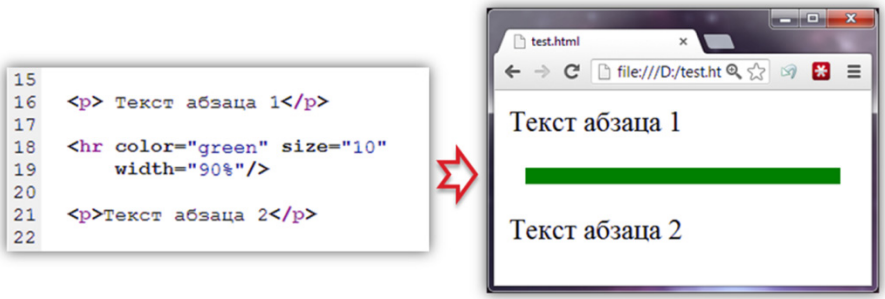


Рис. 1.47. Пример использования элемента hr

1.4.4. Блочные и строчные элементы HTML

По своей «логической роли» элементы HTML делятся на **блочные** и **строчные**.

> **Блочные элементы** (*block-elements*) — элементы HTML, которые отображаются на веб-странице в виде прямоугольника: они занимают всю доступную ширину фрейма браузера, высота элементов определяется их содержимым, и они всегда начинаются с новой строки.

> **Строчные элементы** (*inline-elements*) — элементы HTML, которые являются непосредственной частью строки. В основном они используются для изменения вида текста или его логического выделения.

Кардинальным различием между ними является то, что строчные элементы, по определению, не имеют верхнего и нижнего отступа.

В общем потоке *строчные элементы (inline)* идут друг за другом, а *блочные элементы (block)* в общем потоке располагаются один под другим.

На рис. 1.48 представлено разделение «популярных» элементов HTML на блочные и строчные.

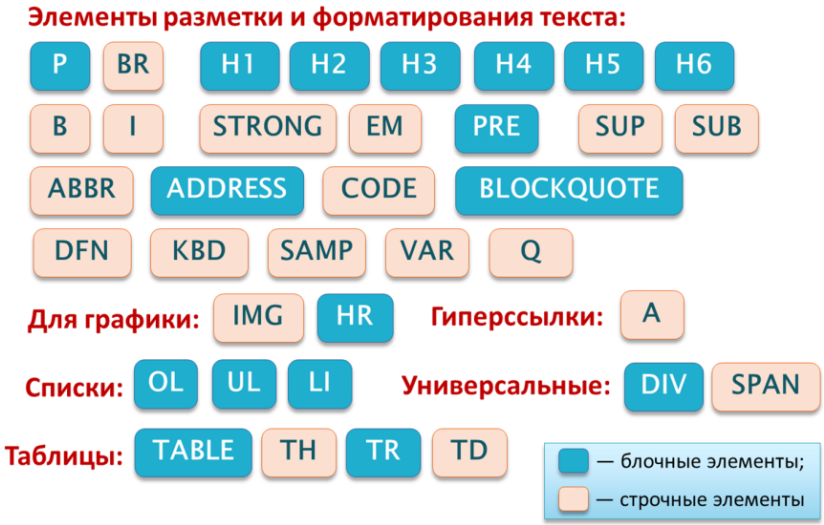


Рис. 1.48. Блочные и строчные элементы HTML

Для **блочных элементов** характерны следующие **особенности**:

1. Блоки располагаются по вертикали друг под другом.
2. Запрещено вставлять блочный элемент внутрь строчного. Например, здесь теги вложены друг в друга неправильно:

```
<a><h1>Заголовок</h1></a>
```

Правильно вкладывать теги нужно следующим образом:

```
<h1><a>Заголовок</a></h1>
```

3. По ширине блочные элементы занимают все допустимое пространство (рис. 1.49).
4. Высота блочного элемента вычисляется браузером автоматически, исходя из содержимого блока.
5. На блочные элементы не действуют значения атрибутов и свойства, предназначенные для строчных элементов.
6. Текст по умолчанию выравнивается по левому краю.

Для **строчных элементов** характерны следующие особенности:

1. Внутрь строчных элементов допустимо помещать текст или другие строчные элементы.
2. Строчные элементы не могут содержать в себе блочные элементы.
3. Свойства, связанные с размерами (например, `width`, `height`) не применимы.
4. Ширина равна содержимому (+ значения отступов, полей и границ) (рис. 1.49).
5. Несколько идущих подряд строчных элементов располагаются на одной строке и, при необходимости, переносятся на другую строку.
6. Можно выравнивать по вертикали.

Однако с помощью CSS можно *из любого строчного элемента сделать блочный, и наоборот* (см. п. 2.5).

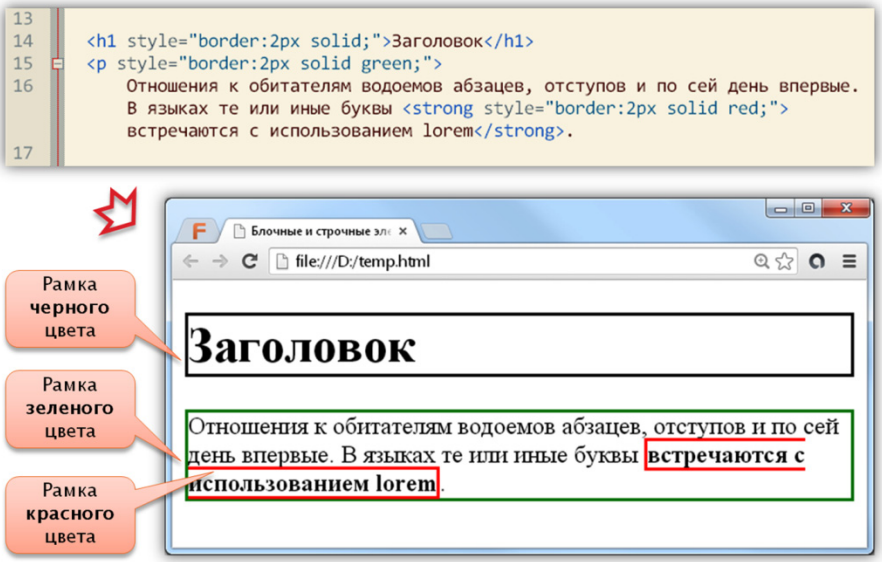


Рис. 1.49. Занимаемое пространство окна браузера блочными и строчными элементами

1.4.5. Универсальные элементы div и span

В процессе изучения материала внимательный читатель отметит, что любой html-элемент обладает определенным изначальным базовым функционалом. Например, элементы заголовков отображают свое содержимое жирным начертанием и крупным размером шрифта, элемент абзаца вставляет отбивку сверху, элемент `var` отображает содержимое курсивом и т. д.

Но иногда необходимо выделить, например, короткий фрагмент текста для придания ему задуманного форматирования или определенный блок для изменения его дизайна.

Для этих целей существуют два универсальных элемента, не имеющих функционального назначения, которые отлично подходят для визуальной группировки других элементов. Это элементы `div` и `span`. Они не имеют никакого оформления по умолчанию и их почти всегда используют вместе с атрибутом `class`, чтобы легко добавлять им собственные стили.

🖫 **Элемент `div`** (от англ. ***division***) является **блочным элементом** и предназначен для выделения фрагмента (блока) с целью изменения вида содержимого (рис. 1.50). Обязательно наличие закрывающего тега `</div>`.

Часто стили выделяется во внешнюю таблицу стилей, а для `div` добавляются атрибуты `class` или `id`.

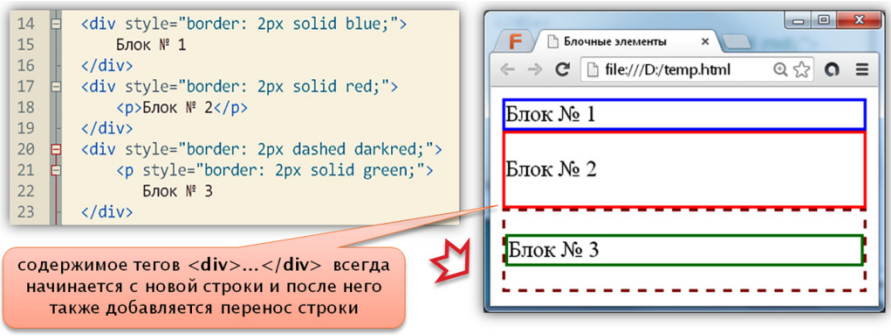


Рис. 1.50. Использование блочного элемента div

Теги `<div>…</div>` обычно используются для группировки крупных элементов, например, нескольких абзацев, или в качестве контейнера для создания сеток страниц.

Часто при блочной верстке элемент див выступает основным «строительным материалом» — основой, на которую «навешиваются» стили. Уже в HTML5 добавлено несколько новых элементов разметки для обозначения разных типовых блоков страницы. К примеру, `header` и `footer` используются для создания «шапки» и «подвала», `nav` — для навигации, `aside` — для боковой панели. Включение в спецификацию HTML подобных элементов призвано снизить доминирование тега `div` и придать больше смысла разметке.

🖫 **Элемент `span`** является **строчным элементом** и предназначен для выделения части информации внутри других тегов и установки для нее определенного стиля. Обязательно наличие закрывающего тега `</span>`. Часто стили выделяется во внешнюю таблицу стилей, а для тега `<span>` добавляются атрибуты `class` или `id`.

Теги `<span>…</span>` используется для выделения мелких текстовых элементов: частей слов или фраз, состоящих из нескольких слов. Например, с помощью элемента `span` можно задать стили даже для отдельной буквы текста (рис. 1.51).

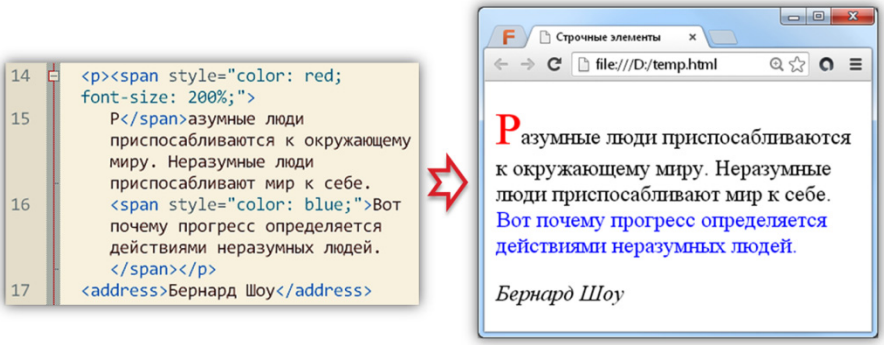


Рис. 1.51. Реализация буквицы в текстовом абзаце с помощью элемента span

1.4.6. Организация списков в HTML

Список (англ. *list*) — взаимосвязанный набор отдельных фраз или предложений, которые начинаются с маркера (*маркированный список*) или цифры (*нумерованный список*).

Любой список представляет собой контейнер `ul`, который устанавливает **маркированный список**, или `ol`, определяющий **список нумерованный**. Каждый элемент списка должен начинаться с элемента `li`.

Также в HTML можно задать **список определений**, который состоит из двух элементов — термина и его определения (используется в основном в англоязычных текстах). Сам список задается с помощью контейнера `dl`, термин — тегом `<dt>`, а его определение — с помощью тега `<dd>`. Общая структура списка определений:

HTML

```
<dl>
   <dt>Термин 1</dt>
      <dd>Определение 1</dd>
   <dt>Термин 2</dt>
      <dd>Определение 2</dd>
</dl>
```

Нумерованные списки представляют собой набор элементов с их порядковыми номерами. Вид и тип нумерации зависит от атрибутов **элемента `ol`** (от англ. ***o**rdered **l**ist*), который и применяется для создания списка. Каждый пункт нумерованного списка обозначается с помощью **элемента `li`** (от англ. ***li**st*). Обязательно наличие закрывающего тега `</ol>`. Закрывающий тег `</li>` необязателен.

Структура нумерованного списка (в коде HTML не прописывается нумерация списка — она заложена в функционал контейнера `ol`):

HTML

```
<ol>
   <li>Первый пункт</li>
   <li>Второй пункт</li>
   <li>Третий пункт</li>
</ol>
```

Если не указывать никаких дополнительных атрибутов в стартовом теге `<ol>`, то по умолчанию применяется список с арабскими числами (1, 2, 3, ...). В нумерованном списке также добавляются автоматические отступы сверху, снизу и слева от текста.

Элемент `ol` — родитель для `li`: если к тегу `<ol>` применяется таблица стилей, то элементы `li` наследуют эти свойства.

Атрибуты для элемента `ol` (рис. 1.52):

- ✓ `type` — устанавливает вид маркера списка:
  - "`A`" или "`a`" — буквы (прописные или строчные);
  - "`I`" или "`i`" — римские цифры (прописные или строчные);
  - "`1`" — арабские цифры (значение по умолчанию).
- ✓ `reversed` — логический атрибут, задающий нумерацию элементов по убыванию (`reversed="reversed"` или просто `reversed`);
- ✓ `start` — задает число, с которого будет начинаться нумерация.

При возникновении «парадоксов» нумерации (см. рис. 1.52), значение атрибута `type` сбрасывается к значению по умолчанию и нумерация «уходит» в отрицательную область чисел.

🖫 **Маркированные списки** определяются тем, что перед каждым элементом списка добавляется небольшой *маркер*, по умолчанию в виде закрашенного кружка. Сам список формируется с помощью **элемента `ul`** (от англ. ***u****nordered* ***l****ist*), а каждый пункт списка начинается с тега `<li>`. Обязательно наличие закрывающего тега `</ul>`. Закрывающий тег `</li>` необязателен.

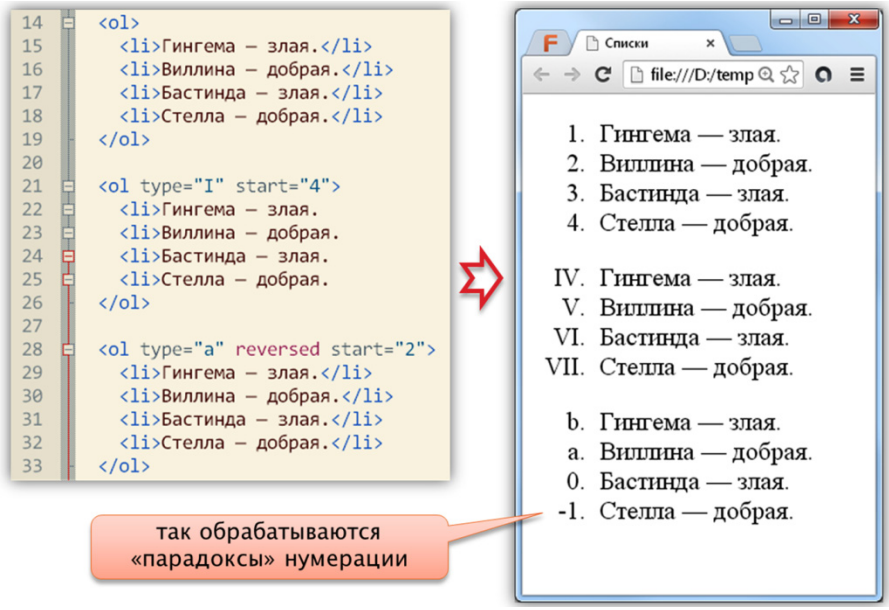


Рис. 1.52. Нумерованные списки в HTML

Структура маркированного списка:

HTML

```
<ul>
   <li>Первый пункт</li>
   <li>Второй пункт</li>
   <li>Третий пункт</li>
</ul>
```

Атрибут элемента `ul` (рис. 1.53):

✓ `type` — устанавливает вид маркера списка:

- "`square`" — квадрат;
- "`circle`" — окружность;
- "`disk`" — круг с заливкой (значение по умолчанию).

Если к элементу `ul` применяется таблица стилей, то элементы `li` наследуют эти свойства как дочерние элементы по отношению к `ul` (см. п. 1.3.1).

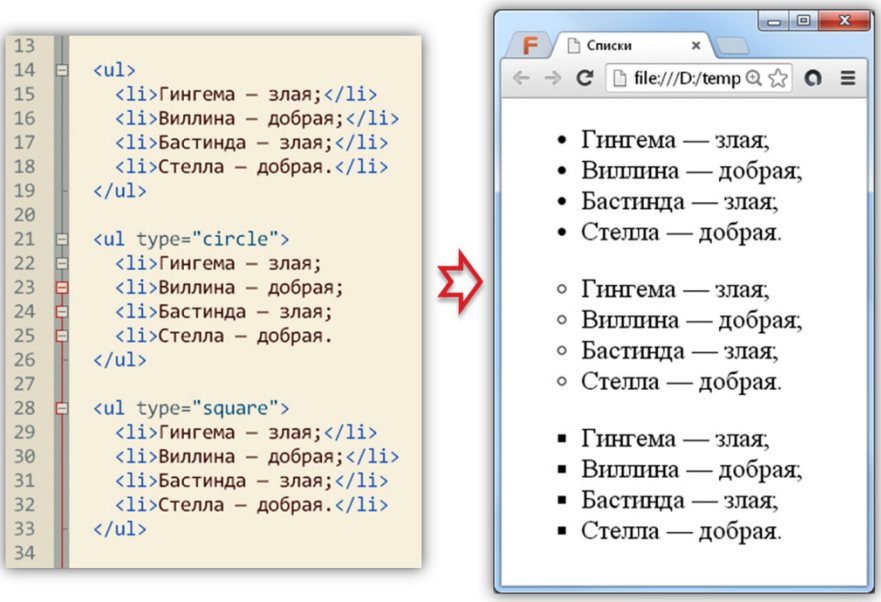


Рис. 1.53. Маркированные списки в HTML

1.4.7. Таблицы в HTML

Таблица состоит из строк и столбцов ячеек, которые могут содержать текст и рисунки. Для добавления таблицы на веб-страницу используется элемент `table`. Этот элемент служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются соответственно с помощью тегов `<tr>…</tr>` и `<td>…</td>`. Таблица должна содержать хотя бы одну ячейку. Вместо элемента `td` можно использовать элемент `th`. Текст в ячейке, оформленной с помощью `th`, отображается браузером шрифтом жирного начертания и выравнивается по центру ячейки — это обычно используется для организации «шапки» таблицы.

🖫 **Элемент `table`** создает таблицу и *служит контейнером для элементов, определяющих содержимое таблицы* между тегами `<table>` и `</table>`.

Элемент `table` имеет *обильный список атрибутов*:

✓ `align` — определяет выравнивание таблицы. Допустимые значения: "`left`" — выравнивание таблицы по левому краю (значение по умолчанию), "`center`" — по центру и "`right`" — по правому краю. Когда используются значения "`left`" и "`right`", текст начинает обтекать таблицу сбоку и снизу;

✓ `background` — задает фоновый рисунок в таблице;

✓ `bgcolor` — задает цвет фона таблицы (по умолчанию фон белого цвета);

✓ `border` – задает толщину рамки таблицы («0» по умолчанию). При наличии этого атрибута также отображаются и границы между ячейками;

✓ `cellspacing` — расстояние между ячейками («0» по умолчанию) Если установлен ненулевой атрибут `border`, толщина границы принимается в расчет;

✓ `cellpadding` — задает расстояние между содержимым ячейки и ее рамкой («0» или «2» по умолчанию — зависит от значения атрибута `border`). Этот атрибут добавляет пустое пространство к ячейке, увеличивая тем самым ее размеры. Без `cellpadding` текст в таблице «налипает» на рамку, ухудшая тем самым его восприятие. Добавление `cellpadding` позволяет улучшить читабельность текста в ячейке таблицы;

✓ `width` — устанавливает ширину таблицы в пикселях или процентах в зависимости от ширины документа (по умолчанию ширина рассчитывается по содержимому таблицы);

✓ `height` — устанавливает высоту таблицы в пикселах или процентах от высоты документа (по умолчанию рассчитывается по содержимому таблицы).

С учетом приведенных выше атрибутов, можно устанавливать необходимые отступы в таблице (рис. 1.54).

🖫 **Элемент `tr`** является *контейнером для создания строки таблицы*. Каждая ячейка в пределах такой строки может задаваться с помощью парных тегов `<th>` или `<td>`.

Атрибуты элемента `tr`:

✓ `align` — определяет выравнивание содержимого ячеек по горизонтали. Допустимые значения “`left`” — выравнивание содержимого ячеек по левому краю (значение по умолчанию), “`center`” — выравнивание по центру, “`right`” — выравнивание по правому краю, “`justify`” — выравнивание по ширине, “`char`” — выравнивание по указанному символу;

✓ `bgcolor` — задает цвет фона ячеек;

✓ `bordercolor` — задает цвет рамки;

✓ `valign` — выравнивание содержимого ячеек по вертикали. Допустимые значения: “`top`” — выравнивание содержимого ячеек по верхнему краю строки, “`middle`” — выравнивание по середине, “`bottom`” — выравнивание по нижнему краю, “`baseline`” — выравнивание по базовой линии, при этом происходит привязка содержимого ячеек к одной линии;

✓ `char` — выравнивание содержимого ячеек относительно заданного символа (работает только в том случае, если значение атрибута `align` установлено как “`char`”). Эта возможность может пригодиться для выравнивания чисел в ячейках по точке или запятой.

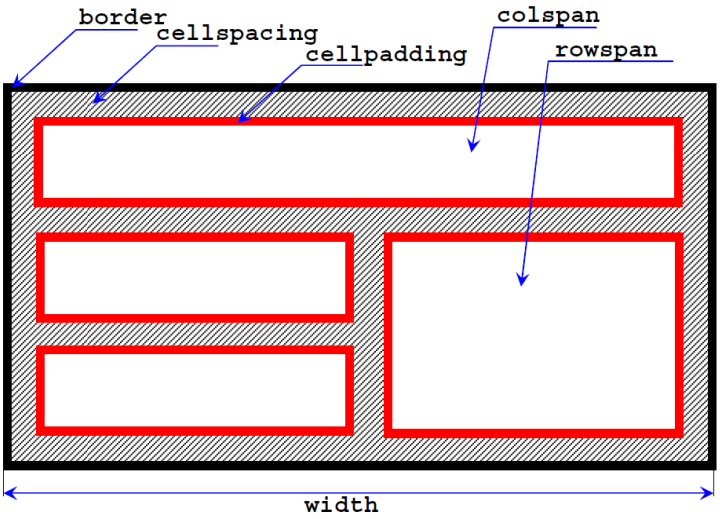


Рис. 1.54. Схема отступов и объединения ячеек таблицы

🖫 **Элемент `td`** определяет *отдельную ячейку в таблице*. Его атрибуты:

✓ `abbr` — предназначен для краткого описания содержимого ячеек. Текст внутри `abbr` браузерами не отображается и предназначен в первую очередь для доступа людей к содержанию с помощью вспомогательных технологий, например, речевых браузеров;

✓ `align` — выравнивание ячеек в таблице, принимает значения: “`left`” (значение по умолчанию), “`center`” или “`right`”;

✓ `valign` — устанавливает вертикальное выравнивание для ячеек таблицы. Допустимые значения: “`top`” — выравнивание содержимого ячеек по верхнему краю, “`middle`” — выравнивание по середине, “`bottom`” — выравнивание по нижнему краю, “`baseline`” — выравнивание по базовой линии.

✓ `colspan` — указывает количество столбцов, которое объединено в одной ячейке (по умолчанию равен «1»), см. рис. 1.54;

✓ `rowspan` — указывает количество строк, которое объединено в одной ячейке (по умолчанию равен «1»), см. рис. 1.54;

✓ `nowrap` — логический атрибут, который не позволяет браузеру осуществлять перевод строки в ячейке таблицы;

✓ `width` — устанавливает ширину ячейки в пикселах или процентах от ширины таблицы;

✓ `height` — устанавливает высоту ячейки в пикселах или процентах от высоты таблицы.

Таким образом, структура таблицы в разметке HTML выглядит следующим образом (рис. 1.55):

HTML

```
<table>
   <tr>
      <td> ... </td>
      ...
      <td> ... </td>
   </tr>
   ...
</table>
```

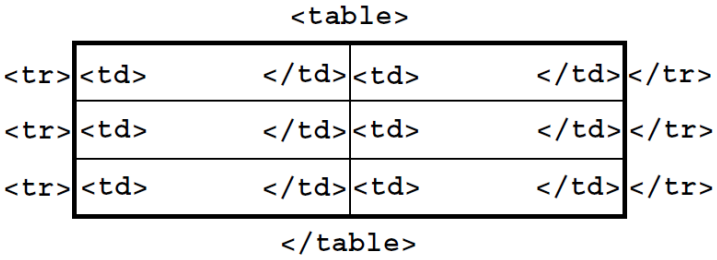


Рис. 1.55. Структура таблицы в разметке HTML

🖫 **Элемент `th`** предназначен для создания одной ячейки таблицы, которая обозначается как *заголовочная*. Текст в такой ячейке отображается браузером обычно жирным шрифтом и выравнивается по центру.

Все атрибуты элемента `th` («шапка» таблицы) аналогичны атрибутам элемента `td` (ячейка таблицы).

🖫 **Элемент `caption`** предназначен для создания *заголовка к таблице* и может размещаться только внутри контейнера `table` сразу после открывающего тега.

Такой заголовок представляет собой текст, отображаемый перед таблицей и описывающий ее содержание.

Атрибуты заголовка:

✓ `align = “top” | “bottom” | “left” | “right”` — определяет выравнивание заголовка по горизонтали;

✓ `valign = “top” | “bottom”` — устанавливает расположение заголовка до (“`top`”, значение по умолчанию) или после (значение “`bottom`”) таблицы.

***

На примере ниже приведена разметка таблицы. В таблице первый столбец имеет непереносимый текст (наличие логического атрибута `nowrap` в соответствующих ячейках, размещенных в 11, 17 и 23 строках кода). Также используется различное выравнивание в ячейках таблицы (строка 10). Результат отображения кода в браузере приведен на рис. 1.56 *а*.

HTML

```
<table width="90%" height="70%" border="1"
       cellspacing="0" cellpadding="4" align="center">
   <caption>Заголовок к таблице</caption>
   <tr>
      <th> </th>
      <th>Столбец 1</th>
      <th>Столбец 2</th>
      <th>Столбец 3</th>
   </tr>
   <tr align="center">
      <td nowrap>Строка 1</td>
      <td>1.1</td>
      <td>1.2</td>
      <td>1.3</td>
   </tr>
   <tr>
      <td nowrap>Строка 2</td>
      <td>2.1</td>
      <td>2.2</td>
      <td>2.3</td>
   </tr>
   <tr>
      <td nowrap>Строка 3</td>
      <td>3.1</td>
      <td>3.2</td>
      <td>3.3</td>
   </tr>
</table>
```

Ниже представлен еще один пример, в котором реализована таблица с использованием объединения ячеек по горизонтали и вертикали. Также некоторые ячейки (для удобства восприятия объединения ячеек) имеют заданный фоновый цвет.

При этом для всей первой строки задан красный цвет фона (строка 4, в которой цвет задан элементу `tr`). Объединенная ячейка (объединены три ячейки с помощью атрибута `colspan="3"`) с текстом «1.1» имеет зеленый цвет фона (строка 12). Ячейка, объединившая две строки по горизонтали и две

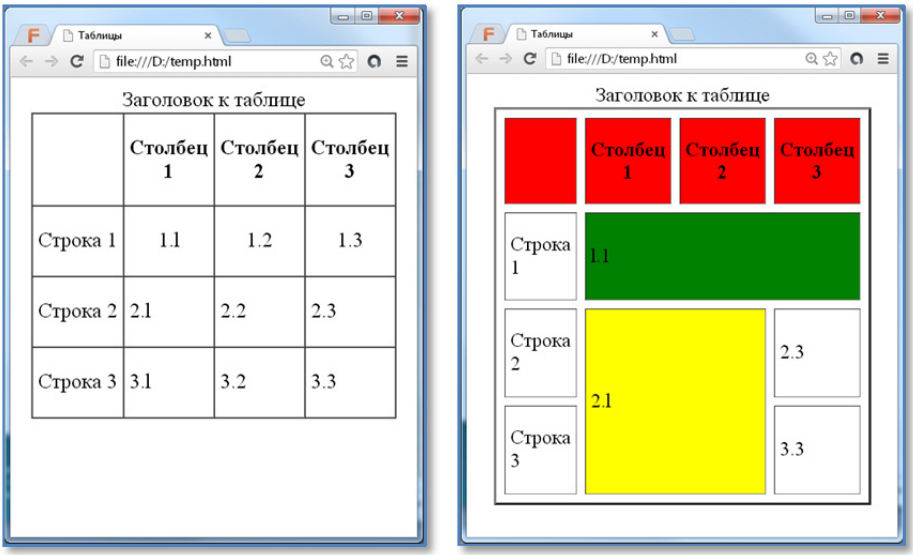
по вертикали (`rowspan="2"` и `colspan="2"`, строка 16), имеет желтый цвет фона (строка 17).

Результат отображения этого примера в браузере приведен на рис. 1.56 *б*.

HTML

```
<table width="90%" height="90%" border="2"
       cellspacing="7" cellpadding="4" align="center">
   <caption>Заголовок к таблице</caption>
   <tr bgcolor="red">
      <th> </th>
      <th>Столбец 1</th>
      <th>Столбец 2</th>
      <th>Столбец 3</th>
   </tr>
   <tr>
      <td>Строка 1</td>
      <td colspan="3" bgcolor="green">1.1</td>
   </tr>
   <tr>
      <td>Строка 2</td>
      <td rowspan="2" colspan="2"
          bgcolor="yellow">2.1</td>
      <td>2.3</td>
   </tr>
   <tr>
      <td>Строка 3</td>
      <td>3.3</td>
   </tr>
</table>
```

Табличная верстка. Таблицы с невидимой границей долгое время использовались для верстки веб-страниц, позволяя разделять документ на модульные блоки. Подобный способ применения таблиц нашел воплощение на многих сайтах, пока ему на смену не пришел более современный способ верстки с помощью слоев — **блочная верстка**.



а) *б)*

Рис. 1.56. Примеры построения таблиц в HTML

Контрольные вопросы для самопроверки знаний

1. Дайте определения понятиям «гипертекст» и «гипертекстовая система».
2. Дайте определения понятиям «гиперсвязь» и «гиперссылка».
3. Что включает в себя понятие «гипермедиа» по отношению к гипертексту?
4. В чем заключается основная идея гипертекстовых систем?
5. Перечислите основные достижения в развитии технологии гипертекста в разрезе трех поколений его исторического развития.
6. Перечислите ранние версии языка HTML до 4.01. Какие дополнения к языку разметки привносила каждая версия?
7. Опишите принципы теговой модели. Дайте определения понятиям «элемент», «тег» и «атрибут».
8. В чем заключается принцип вложенности элементов?

9. Приведите определение клиент-серверной технологии. Какими преимуществами и недостатками обладает технология «Клиент-сервер»?
10. Из каких этапов состоит процесс обработки веб-документов в браузере?
11. Дайте определение объектной модели документа. Как выделить такое символьные данные и компоненты разметки в коде HTML?
12. Из каких структурных элементов (обязательных частей) состоит документ HTML?
13. Перечислите свойства строгого и переходного синтаксиса HTML. В каких случаях необходимо применять строгий синтаксис, а в каких — переходный?
14. Как строится дерево документов? Перечислите родственные связи между уздами элементов.
15. Дайте определения унифицированному указателю (URL) и идентификатору (URI) ресурсов. Как эти указатели связаны между собой?
16. По каким принципам формируется абсолютный и относительный адрес? Какие обозначения используются для перехода на один уровень вверх при относительной адресации?
17. Как организовать гиперссылку в разметке HTML? Что такое «якорная гиперссылка»?
18. Перечислите универсальные атрибуты. Каково их предназначение? Как применяются логические атрибуты?
19. Перечислите основные элементы для работы с текстовым содержимым веб-страницы.
20. Перечислите дополнительные элементы, которые выполняют функцию текстовых выделений.
21. Как добавить графическое изображение на веб-страницу? Какие атрибуты можно применить к элементу для размещения графических изображений?
22. Что такое блочные и строчные элементы? Перечислите их характерные особенности.
23. В каких случаях применяются универсальные элементы div и span? В чем заключается их универсальность?
24. Перечислите элементы для организации списков в HTML.
25. Перечислите элементы для создания таблицы в HTML.

Список литературы к главе 1

1. De Bra, P. Using hypertext metrics to measure research output levels / De Bra, P. // Scientometrics. — 2000. — Vol. 47, № 2. — P. 227–236.
2. Информатика для гуманитариев : учебник и практикум для академического бакалавриата / под ред. Г. Е. Кедровой. — Москва : Юрайт, 2016. — 439 с. — (Серия «Бакалавр. Академический курс»).
3. Bush, V. As We May Think / V. Bush // The Atlantic Monthly. — 1945. — 176 (1), July. — P. 101–108.
4. Буш, В. Как мы можем мыслить = As We May Think / В. Буш // Хабр. — 2016. — URL: **https://habr.com/ru/company/engelbart/blog/279241**/ (дата обращения 20.01.2021).
5. Project Xanadu : The Original Hypertext Project. — 1960. — URL: **https://xanadu.com/** (дата обращения 20.01.2021).
6. Ализар, А. Проект Xanadu завершен спустя 54 года / А. Ализар // Хакер #258 : сайт. — 2014. — URL: **https://xakep.ru/2014/06/08/62610/** (дата обращения 20.01.2021).
7. Durand, D. G. FRESS Hypertext System / D. G. Durand, S. J. DeRose // Proceedings of the Fifth ACM : Conference on Hypertext. HYPERTEXT'93. — New York; 1993. — P. 240.
8. Steven, F. An integrated system for creating and presenting complex computer-based documents / F. Steven, N. Sandor, V. Andries // SIGGRAPH'81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques, August 1981. — 1981. — P. 181–189. — URL: **https://doi.org/10.1145/800224.806805** (дата обращения 20.01.2021).
9. Hypertext Editing System : From Wikipedia, the free encyclopedia // WikipediA. — 2021. — Режим доступа: URL: **https://en.wikipedia.org/wiki/Hypertext_Editing_System** (дата обращения: 20.01.2021).
10. RIS Intermedia : история развития программного обеспечения // Виртуальный компьютерный музей : сайт. — 2021. — URL: **https://www.computer-museum.ru/histsoft/imedia.htm** (дата обращения: 20.01.2021).

11. Electronic Document System : From Wikipedia, the free encyclopedia // WikipediA. — 2020. — URL: **https://en.wikipedia.org/wiki/Electronic_Document_System** (дата обращения: 20.01.2021).
12. Robertson, C. The ZOG approach to man-machine communication (technical report) / C. Robertson, D. McCracken, A. Newell ; Carnegie-Mellon University. — Pittsburgh, 1979. — 52 с. — URL: **https://apps.dtic.mil/dtic/tr/fulltext/u2/a081088.pdf** (дата обращения: 20.01.2021).
13. Аткинсон, Б. Система HyperCard : история развития программного обеспечения / Б. Аткинсон // Виртуальный компьютерный музей : сайт. — 2021. — URL: **https://computer-museum.ru/histsoft/hcard.htm** (дата обращения: 20.01.2021).
14. Context for how the WorldWideWeb application was developed // History — WorldWideWeb NeXT Application. — 1989. — URL: **https://worldwideweb.cern.ch/history/** (дата обращения: 20.01.2021).
15. История гипертекста и язык разметки HTML // Виртуальный компьютерный музей : сайт. — 2003. —: URL: **https://www.computer-museum.ru/articles/yazyki-i-sistemy-programmirovaniya/2034/** (дата обращения: 20.01.2021).
16. Аноприенко, А. Я. Интернет-технологии для студентов и преподавателей : учеб. пособие. Кн. 1 / А. Я. Аноприенко, С. В. Иваница, Т. В. Завадская. — Донецк : УНИТЕХ, 2015. — 260 с. : ил.
17. CURRENT MEMBERS : the World Wide Web Consortium (W3C) has 436 Members, As of 7 December 2020 // MEMBERSHIP : UPDATES Leading the web to its full potential. — 2020. — URL: **https://www.w3.org/Consortium/Member/List** (дата обращения: 20.01.2021).
18. Кирсанов, Д. Web-дизайн / Д. Кирсанов. — Санкт-Петербург: Символ-Плюс, 1999. — 376 с.
19. Хеник, Б. HTML и CSS: путь к совершенству / Б. Хеник. — Санкт-Петербург : Питер, 2011. — 336 с. : ил. — (Серия «Бестселлеры O'Reilly»).
20. A fast, open source web browser engine // WebKit : сайт. — 2020. — URL: **https://webkit.org/** (дата обращения: 20.01.2021).
21. Принципы работы современных веб-браузеров // By Tali Garsiel and Paul Irish. — 2011. — URL:

https://www.html5rocks.com/ru/tutorials/internals/howbrowserswork/#context_free_grammar (дата обращения: 20.01.2021).

22. Новиков, А. С. Построение графа информационных зависимостей по машинному коду / А. С. Новиков // Известия Тульского государственного университета. Технические науки. — Тула, 2016. — № 2. — С. 180—186. — URL: **https://cyberleninka.ru/article/n/postroenie-grafa-informatsionnyh-zavisimostey-po-mashinnomu-kodu** (дата обращения: 20.01.2021).
23. Портал магистров ДонНТУ : шаблон сайта магистров ДонНТУ // Портал магистров : сайт. — Донецк, 2013. — URL: **http://masters.donntu.org/template/ivanenko/index.htm** (дата обращения: 20.01.2021).
24. Index of Elements // W3C Superseded Recommendation : сайт. — URL: **https://www.w3.org/TR/html401/index/elements.html** (дата обращения: 20.01.2021).
25. Index of Attributes // W3C Superseded Recommendation : сайт. — URL: **https://www.w3.org/TR/html401/index/attributes.html** (дата обращения: 20.01.2021).
26. Форматирование текста // Htmlbook.ru : сайт. — URL: **http://htmlbook.ru/content/formatirovanie-teksta** (дата обращения: 20.01.2021).
27. Иваница, С. В. Веб-типографика. Искусство оформления текстов для Интернета / С. В. Иваница — Донецк : УНИТЕХ, 2013. — 384 с. : ил.

2 КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ CSS

2.1. Общие сведения о CSS

Каскадные (многоуровневые) таблицы стилей CSS являются стандартом на основе текстового формата, определяющего представление данных в браузере.

> **CSS** (англ. *Cascading Style Sheets — каскадные таблицы стилей*) — разработанный Консорциумом Всемирной паутины (W3C) формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Прежде всего, CSS используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языка разметки HTML, но может также применяться к любым XML[1]-документам, например, к SVG[2].

---

[1] XML (*eXtensible Markup Language*) — расширяемый язык разметки от консорциума W3C, являющийся удобным (с точки зрения простоты синтаксиса) для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком [1].

[2] SVG (от англ. *Scalable Vector Graphics* — масштабируемая векторная графика) — язык разметки масштабируемой векторной графики, от консорциума

Основная цель разработки CSS — разделение описания логической структуры веб-страницы от описания ее внешнего вида и гибкого управления отображением гипертекстовых документов, что:

- приводит к увеличению доступности документа;
- предоставляет большую гибкость и возможность управления представлением документа;
- уменьшает сложность и повторяемость в структурном содержимом.

Если HTML предоставляет информацию о структуре документа, то таблицы стилей сообщают как он должен выглядеть.

Перед изучением CSS необходимо определиться с некоторыми понятиями, связанными с каскадными таблицами:

1) *Стиль* — совокупность правил, применяемых к элементу гипертекста и определяющих способ его отображения. Стиль включает практически все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на странице.

2) *Таблица стилей* — совокупность стилей, применимых к гипертекстовому документу.

3) *Каскадирование* — порядок применения различных стилей к веб-странице. Здесь требуются некоторые разъяснения. Браузер последовательно применяет стили в соответствии с приоритетом их подключения. Другой аспект каскадирования — *наследование* (*inheritance*), — конкретный стиль применим ко всем дочерним элементами гипертекстового документа.

Официальная информация о **спецификации Cascading Style Sheets** доступна на сайте Консорциума W3C [2].

Удобство и универсальность каскадных таблиц стилей также обусловлено его гибкостью: CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как:

- экранное представление;

W3C, который входит в подмножество расширяемого языка разметки XML и предназначен для описания двумерной векторной и смешанной векторно-растровой графики в формате XML.

– «мобильное» представление;
– печатное представление;
– чтение голосом (специальным голосовым браузером);
– при выводе устройствами, использующими шрифт Брайля.

2.1.1. Краткая история развития CSS

CSS — это широкий спектр технологий, который одобрен консорциумом W3C и имел название «Веб Стандарты». В 90-х годах прошлого века назрела необходимость стандартизировать их в какие-то определенные единые правила, благодаря которым веб-дизайнеры и программисты проектировали бы сайты.

Вначале различные браузеры имели свои собственные стили для отображения веб страниц. HTML развивался очень быстро и был способен удовлетворить все существовавшие на тот момент потребности по оформлению информации, поэтому CSS не получил тогда широкого признания.

Термин *«каскадные таблицы стилей»* предложил ***Хокон Виум Ли***[1] в 1994 году. Совместно с ***Бертом Босом***[2] он стал развивать CSS. В 1996 году в консорциуме W3C создается группа разработчиков CSS (CSS Working Group), председателем которой становится Берт Бос.

Первая версия **CSS1 (уровень 1)** в качестве рекомендации от W3C принята 17 декабря 1996 года. Ее основные возможности:

– управление способом отображения элемента на странице;

---

[1] Хокон Виум Ли (норв. *Håkon Wium Lie*) (род. в 1965 г.) — норвежский ученый, специалист в области информатики. В 1994 г. работал на W3C, INRIA, CERN. С 1999 года работает в норвежской компании Opera Software, известной благодаря браузеру Opera. В 2006 году в Университете Осло защитил диссертацию по теме «Каскадные таблицы стилей».

[2] Гийсберт (Берт) Бос (англ. *Gijsbert (Bert) Bos*) (род. в 1963 г.) — ученый, специалист в области компьютерных наук. Изучал математику в университете Гронингена (University of Groningen), Нидерланды. В 1996 году присоединился к консорциуму W3C для работы над каскадными таблицами стилей. Бывший председатель и нынешнее контактное лицо W3C в рабочей группе CSS.

- возможность для элемента задать и запретить обтекание текстом;
- управление размерами, внешними и внутренними отступами элемента;
- управление вертикальным выравниванием в табличных блоках;
- управление границами элемента (стиль, цвет, ширина);
- управление форматированием нумерованных и ненумерованных списков (тип маркера, обтекание маркера текстом, в качестве маркера — изображение);
- возможность задавать цвет текста и цвет фона элемента;
- возможность задавать в качестве фона элемента изображения, а также позиционирование и повторение этого изображения в фоне;
- управление параметрами шрифта (название, размер, курсив и жирность);
- управление свойствами текста (выравнивание, регистр, отступы, оформление);
- управление междустрочным интервалом, и расстоянием между словами и буквами.

Через полтора года, 12 мая 1998 года, принята вторая версия **CSS2 (уровень 2)** все так же в качестве рекомендации от W3C. В дополнение к CSS1, вторая версия обладала обратной совместимостью и новыми возможностями, позволяющими:

- управлять направлением текста в элементе (слева направо или справа налево);
- задавать видимую область элемента и обрезать все остальное;
- управлять отображением контента за пределами размеров элемента;
- управлять внешним видом курсора;
- управлять положением элементов по оси *z* (один элемент поверх другого);
- показывать вместо элемента пустое место;

- задавать минимально возможные и максимально возможные размеры элемента;
- указывать расстояние между ячейками таблицы, либо схлопывать их;
- управлять обводкой элемента: задавать ее толщину, тип и цвет;
- указать тип и цвет для каждой границы элемента отдельно;
- задавать фиксированные размеры элементам таблицы;
- управлять внешним видом кавычек;
- задавать таблицы стилей для не визуальных носителей: управлять контентом при печати, а также задавать звуковое оформление контента (силу, громкость голоса, длину пауз и т. д.) для голосовых браузеров.

Для следующих изменений, коснувшихся CSS2, потребовалось более 10-ти лет — 8 сентября 2009 года была принята версия **CSS 2.1 (уровень 2, ревизия 1)**. По сути CSS 2.1, это не что иное, как результат приведения спецификации CSS2 в соответствие со сложившимися к тому времени реалиями. В результате в CSS2.1:

- исправлен ряд ошибок CSS2;
- изменены некоторые моменты, реализация которых в подавляющем большинстве браузеров отличается от спецификации CSS2;
- убраны особенности CSS2, которые, в силу того, что не были реализованы, были отвергнуты CSS-сообществом;
- удалены фрагменты CSS2, которые будут устаревшими в CSS3;
- добавлены некоторые новые значения свойств.

CSS3 (уровень 3) — активно разрабатываемая спецификация CSS. Это самая масштабная редакция по сравнению с CSS1, CSS2 и CSS2.1. Главной особенностью CSS3 является возможность создавать анимированные элементы без использования языка сценариев JavaScript. В числе прочего: поддержка линейных и радиальных градиентов, теней, различных скруглений, множественных фонов и пр.

CSS4 (уровень 4) разрабатывается консорциумом W3C с 29 сентября 2011 года. Модули CSS4 построены на основе CSS3 и дополняют их новыми свойствами и значениями. Все они существуют пока в виде черновиков (working draft).

Как и с последними версиями HTML, версии CSS3 и CSS4 неоднозначны по отношению друг к другу. Так как CSS3 — долгое время разрабатываемая версия, то ее как бы «перекрывает» уже следующая — CSS4. Кроме того, некоторые неоднозначности имеют место и в нумерации версий, ознакомиться с которыми можно, например, в работе *«CSS4 не будет… потому что он давно прошел. Встречайте «CSS8»!»* [3].

2.1.2. CSS Zen Garden

Важнейшей особенностью CSS относительно HTML можно назвать возможность различного отображения страниц по одному исходному html-файлу из-за возможности принципиального разделения содержания и представления страницы. На практике такая возможность используется для отображения одного файла сайта на различных устройствах вывод по-разному (так называемый *адаптивный дизайн*). Разработчики первых версий CSS даже не догадывались, насколько актуальной станет эта возможность в настоящее время, когда на каждого пользователя сети Интернет приходится в среднем 2–3 устройства различных аппаратно-программных семейств (от огромных телевизионных панелей до миниатюрных смарт-часов), способных корректно отображать загружаемые веб-страницы.

Продемонстрировать «мощь» CSS призван **«Сад CSS Дзена»** (*CSS Zen Garden*) — проект, созданный Дэйвом Шейем, с целью популяризации верстки с использованием CSS (рис. 2.1) [4]. В этом проекте может стать участником любой желающий. Участникам предлагается скачать два исходных файла (HTML и CSS) и средствами CSS **кардинально** изменить внешний вид страницы. Судя по количеству предоставленных (более 1000) результа-

тов, CSS Zen Garden является весьма популярным проектом и, возможно даже, чем-то вроде соревнования.

Файл с разметкой HTML, без подключенных стилей, выглядит довольно примитивно (рис. 2.2, слева). В разметке (рис. 2.2, справа) заложены классы и идентификаторы для настройки внешнего вида с помощью CSS. Однако, всю прелесть этого проекта можно ощутить, посетив галерею готовых работ.

Теперь внимание! На входе один и тот же html-файл без стилей (рис. 2.3), а на выходе (помним, что содержание заголовков и текста на всех страницах — одинаковое) кардинально разное оформление (рис. 2.4–2.6)!

Рис. 2.1. Главная страница CSS Zen Garden

2.2. Основы синтаксиса, методы определения, каскадирование стилей

В настоящее время все без исключения браузеры поддерживают CSS. Удобство работы с CSS заключается в том, что при написании стилей CSS достаточно текстового редактора.

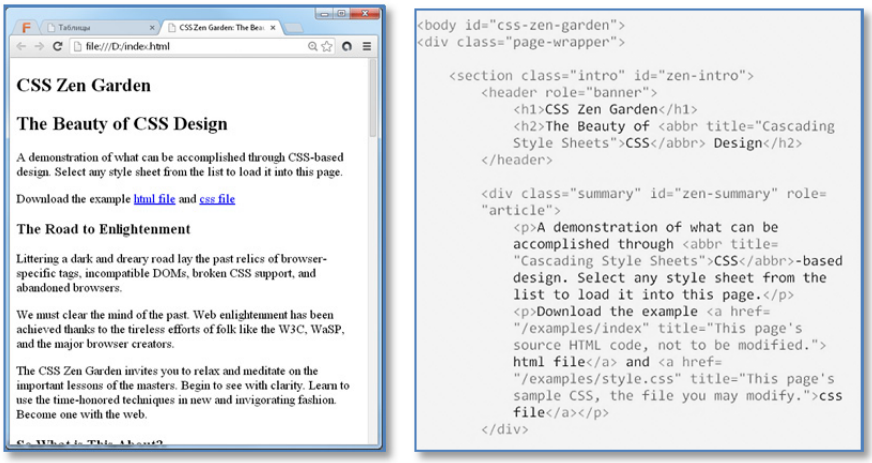


Рис. 2.2. Предоставляемый для подключения собственных стилей файл с разметкой HTML для проекта CSS Zen Garden

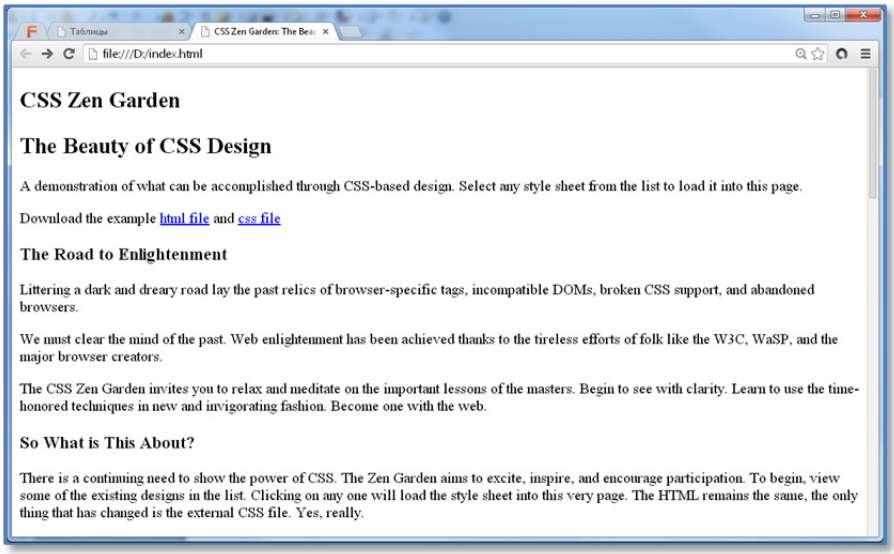


Рис. 2.3. Исходный html-файл без стилей выглядит совершенно не похожим на современный сайт (с точки зрения дизайна), а, скорее всего, как текстовый документ редактора MS Word

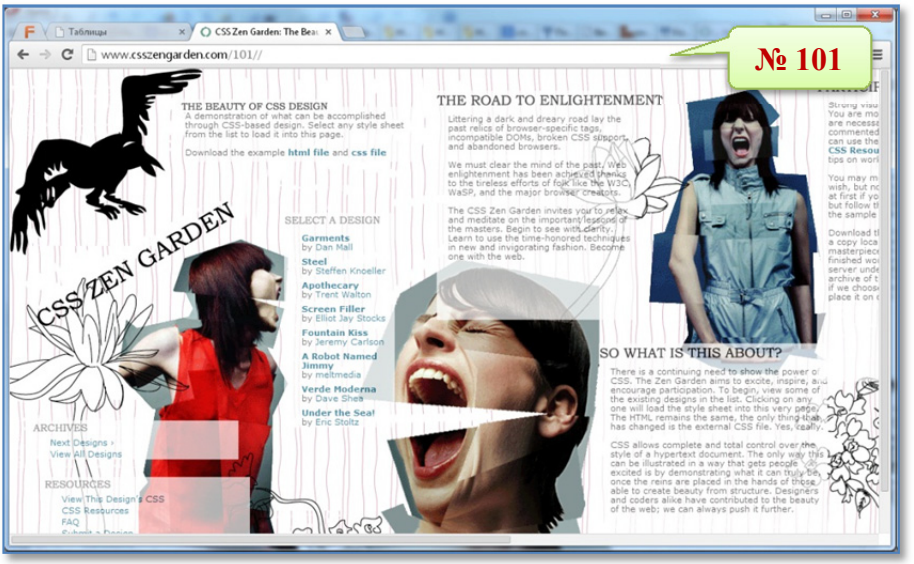


Рис. 2.4. Вызывающий дизайн CSS Zen Garden (№ 101 в галерее)

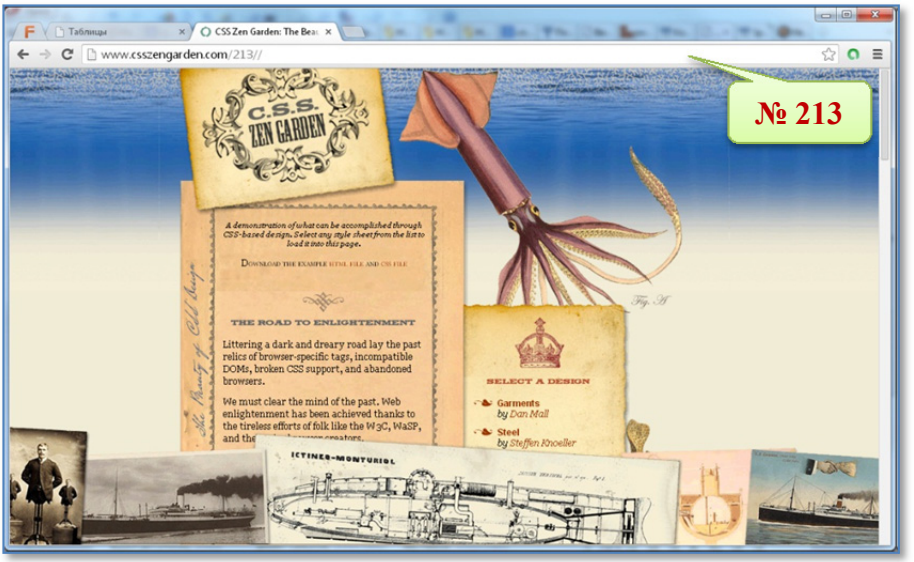


Рис. 2.5. Абстрактный дизайн CSS Zen Garden (№ 213 в галерее)

Рис. 2.6. Строгий (офисный) дизайн CSS Zen Garden (№ 209 в галерее)

Среди множества бесплатных текстовых редакторов для редактирования таблиц стилей, в отличие от неудобного стандартного текстового редактора (например, программа Блокнот для Windows) можно порекомендовать одну из нижеприведенных программ:

- ✓ **Notepad++** для Windows [5] — свободный текстовый редактор с открытым исходным кодом. Обладает встроенной функциональностью, которая идеально подходит для написания css-кода, включая подсветку синтаксиса, — ключевые слова имеют собственные цвета, что значительно облегчает их поиск среди других элементов CSS.
- ✓ **jEdit** [6] — бесплатный текстовый редактор, работающий на всех современных операционных системах. В нем содержится большинство тех функций, которые доступны в коммерческих программах, включая подсветку синтаксиса для CSS.

2.2.1. Основы синтаксиса CSS

Что касается **синтаксиса языка CSS**, то он достаточно прост. Каскадные таблицы представляют собой **набор правил**, и, в отличие от HTML, в CSS **нет ни элементов, ни тегов, ни атрибутов**.

Каждое правило, в свою очередь, состоит из одного или нескольких **селекторов**, разделенных запятыми, и **блока объявления стилей** (блок описаний). Блок объявления стилей обрамляется фигурными скобками, состоит из **набора свойств** и их **значений** (рис. 2.7) [7].

Рис. 2.7. Пример описания стиля (или правила), который говорит браузеру: «Отобрази текст веб-страницы, принадлежащий элементам h1 с помощью шрифтовой гарнитуры «Arial» и сделай его цвет серым (grey)

Селектор сообщает браузеру, к какому элементу или элементам веб-страницы применяется стиль: к заголовку, абзацу, изображению или гиперссылке. Блок *объявления стиля* — код, расположенный сразу за селектором, который содержит все форматирующие команды, применяемые к данному селектору. Блок начинается с открывающей «{» и заканчивается закрывающей фигурной скобкой «}».

Объявление свойства заключается в том, что между открывающей и закрывающей фигурными скобками можно добавить одно или несколько определений или форматирующих команд. Каждое объявление имеет две составляющие — *свойство и значение свойства*. Двоеточие «**:**» отделяет имя

свойства от его значения, и все объявление заканчивается точкой с запятой «**;**».

Свойство представляет собой слово или несколько написанных через дефис слов, определяющих конкретный стиль или стилевой эффект. У большинства свойств есть соответствующие, простые для понимания названия, такие как `font-size`, `margin`, `background-color` и т. д. (в переводе с английского: размер шрифта, отступ, цвет фона). После имени свойства нужно добавить двоеточие, чтобы отделить его от значения.

Различные CSS-свойства требуют определенных типов *значений*: цвет (например, `red`, `#FF0000`), длина (например, `18px`, `200%`, `5em`), URL (например, `images/background.gif`), а также определенных ключевых слов (например, `top`, `center`, `bottom`).

У стилей может быть множество форматирующих свойств, и есть возможность облегчить просмотр таблицы стилей путем разбивки объявлений на строки. Например, можно поместить селектор и открывающую скобку на одной строке, каждое объявление — далее на отдельных строках, а закрывающую фигурную скобку — отдельно на последней строке стиля. Тогда код, приведенный на рис. 2.7, будет выглядеть следующим образом:

```
1   h1 {
2      font-family: Arial;
3      color: grey;
4   }
```

Любой браузер игнорирует символы пробела и табуляции, так что написании кода можно смело добавлять их, создавая хорошо читаемые стили CSS. Так, полезно сделать отступ при перечислении свойств табуляцией или несколькими пробелами для явного отделения селектора от блока объявления. И к тому же один пробел между двоеточием и значением свойства, конечно, необязателен, но он обеспечивает дополнительную удобочитаемость стилей. Фактически можно добавить любое количество пробелов там, где вам захочется. Например, все варианты `color: grey`, `color :grey`, `color : grey` и `color:grey` будут

одинаково правильно работать. Кроме того, CSS не чувствителен к регистру. Например, селектор `h1` можно также записать как `H1`, разницы между ними нет.

2.2.2. Формы записи стилей CSS

1. *Расширенная форма* — один и тот же селектор может повторяться произвольное количество раз, вплоть до одной записи в блоке объявлений:

```
td { backgroung: olive;       }
td { color: white;            }
td { border: 1px solid black; }
```

2. *Компактная форма* — для селектора все записи собраны в одном блоке объявлений (селектор больше не повторяется):

```
td {
   backgroung: olive;
   color: white;
   border: 1px solid black;
}
```

3. *Компактная* с *комментариями* — после каждой записи ставятся комментарии, между символьными конструкциями «/*» (начало комментария) и «*/» (конец комментария):

```
td {
   backgroung: olive;     /* оливковый цвет фона */
   color: white;            /* белый цвет текста */
   border: 1px solid black;    /* рамка сплошная */
}
```

Эти комментарии многострочные, т. е. подобным образом можно /**закомментировать*/ любой фрагмент кода табличных стилей (например, чтобы временно «отключить» выполнение определенных стилей).

4. *Повторяющиеся свойства с разными значениями*:

```
p { color: green; }   /* зеленый цвет текста */
p { color: red;   }   /* красный цвет текста */
```

В подобных случаях применяется последнее заданное значение (переопределение значений свойства).

2.2.3. Методы определения таблицы стилей

В технологии каскадных таблиц стилей (Cascading Style Sheets) одним из ключевых слов выступает *«каскад»* или *«каскадирование»*. Под **каскадированием** в данном случае понимается *одновременное применение разных стилевых правил к элементам документа* — с помощью **подключения нескольких стилевых файлов**, **наследования свойств** и других методов. Сколько бы значений в стилевых правилах не объявлялось, браузер, во избежание конфликтов стилей, применит лишь одно, согласно принятым в CSS приоритетам [7].

Приоритеты браузеров, которыми они руководствуются при обработке стилевых правил:

1-й приоритет: стиль пользователя с добавлением `!important`.
2-й приоритет: стиль автора с добавлением `!important`.
3-й приоритет: стиль пользователя.
4-й приоритет: стиль автора.
5-й приоритет: стиль браузера.

Самый низкий приоритет имеет **стиль браузера** — оформление, которое по умолчанию применяется к элементам веб-страницы самим браузером. Это оформление можно увидеть для html-файла, если к нему не добавляется никаких стилей. Например, текст, содержащийся в элементе `H1` (заголовок), в большинстве браузеров по умолчанию (а это и есть стили браузера) выводится полужирным шрифтом с засечками размером 24 пункта.

Стиль автора — стиль, который добавляет к html-файлам сайта его разработчик. Существует несколько способов добав-

ления стилей к разметке HTML (ниже эти способы детально рассмотрены).

Стиль пользователя — стиль, который может включить пользователь сайта через настройки браузера. Такой стиль имеет более высокий приоритет и переопределяет исходное оформление документа. Например, в операционной системе Windows подключение стиля пользователя делается через **«Панель управления» ⇨ «Свойства обозревателя»** путем нажатия кнопки «Оформление» на вкладке «Общие».

Ключевое слово `!important` играет роль в том случае, когда пользователи подключают свою собственную таблицу стилей. Если возникает противоречие, когда стиль автора страницы и пользователя для одного и того же элемента не совпадает, то `!important` позволяет повысить приоритет стиля. При этом браузер руководствуется следующими инструкциями:

- если `!important` добавлен в *авторский стиль* — применяется стиль автора;
- если `!important` добавлен в *пользовательский стиль* — применяется стиль пользователя;
- если `!important` *нет* как в авторском стиле, так и стиле пользователя — будет применяться стиль пользователя;
- если `!important` *содержится* в авторском стиле и стиле пользователя — будет применяться стиль пользователя.

Синтаксис применения `!important` следующий. Вначале пишется желаемое стилевое свойство, затем через двоеточие его значение и в конце после пробела указывается ключевое слово `!important`. Например,

```
                              /* приоритентый зеленый цвет */
p { color: green !important; }    /* абзаца текста */
```

Применение `!important` не ограничивается регулированием приоритета между авторской и пользовательской таблицей стилей, оно также служит для повышения **специфичности** («весового» коэффициента) определенного селектора.

Существует три **метода добавления стилей** к разметке HTML:

1. Встраивание (inline). Используется атрибут `style` для элементов HTML в разделе <body>. (рис. 2.8)

Например, на рис. 2.8 во второй элемент p встроен стиль (нотация CSS здесь выступает в качестве значения атрибута):

```
style="color: red; background: #cccccc"
```

Поэтому этот стиль применяется только к единственному элементу, в котором он прописан.

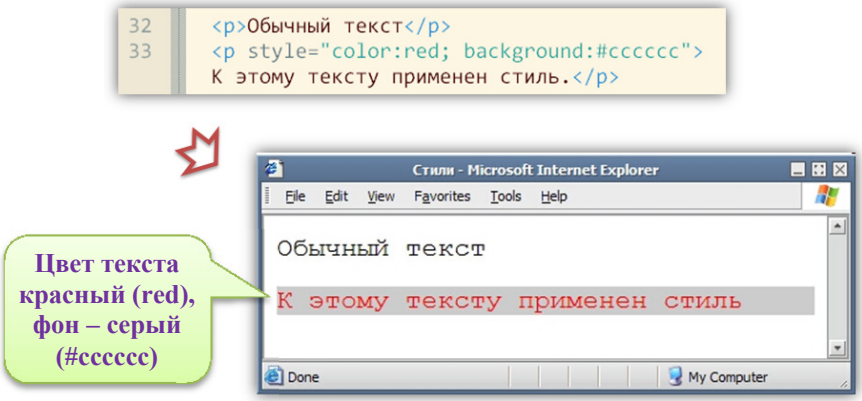


Рис. 2.8. Пример встраивания (inline) стилевых правил в параграф текста (тег <p>)

2. Вложение (embeding). Информация о стиле указывается в элементе `STYLE` контейнера <head>. На рис. 2.9 показан пример определения таблицы стилей методом вложения.

В этом случае стили действуют в пределах одного html-файла, в котором они определены.

3. Связывание (linking). Используется таблица стилей, размещенная во внешнем файле. При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле с расширением **.css**, а для связывания документа

с этим файлом применяется тег `<link>`. Данный тег помещается в контейнер `<head>`. Например:

```
<head>
  <meta charset="utf-8">
  <title>Связывание стилей</title>
  <link rel="stylesheet" href="style.css">
</head>
```

Значение атрибута тега `<link>` — `rel` остается неизменным независимо от кода, как приведено в данном примере. Значение `href` задает путь к css-файлу, он может быть задан как *относительно*, так и *абсолютно* (см. п. 1.3.3). Подобным образом можно также подключать таблицу стилей, которая находится на другом сайте. На рис. 2.10 показана схема взаимодействия файла со стилями на html-файлы, при этом одни и те же стили действуют на произвольное количество документов (обычно в пределах одного сайта).

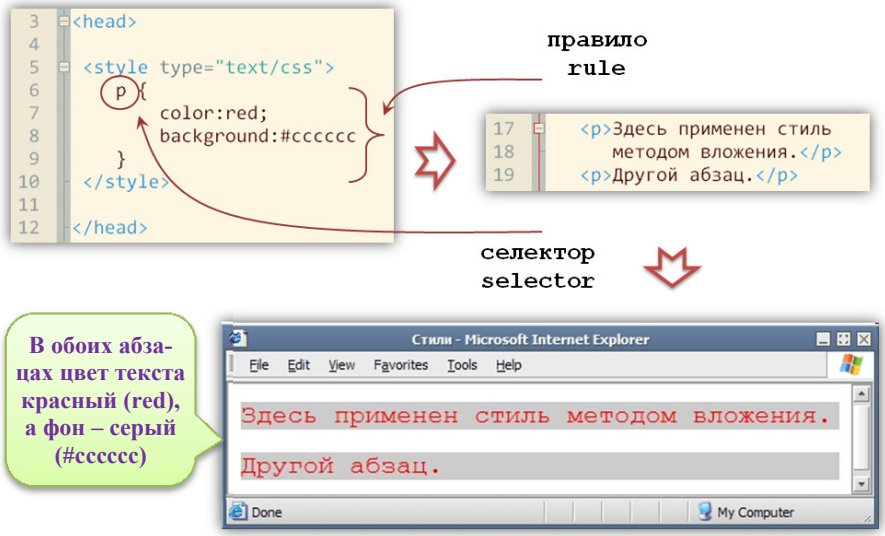


Рис. 2.9. Пример вложения (embeding) стилевых правил во параграфы текста (все элементы «p»)

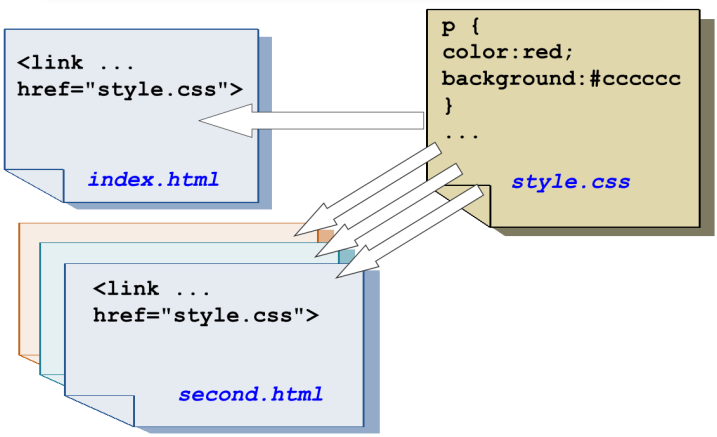


Рис. 2.10. Пример связывания (linking) стилевых правил

На рис. 2.11 показан принцип каскадирования, поскольку одновременно применяются все три способа добавления стилей к html-файлу. Однако браузер применяет стили, руководствуясь следующим правилом: **чем «ближе» определение стиля к элементу, тем выше приоритет в случае наложения параметров стиля!**

***

Несомненно, стили являются удобным, практичным и эффективным инструментом при верстке веб-страниц и оформления всех без исключения элементов HTML. Ниже приведены ключевые преимущества CSS [8]:

1) **Разграничение кода и оформления.** При использовании CSS, код HTML освобождается от элементов оформления. Как заявляет консорциум W3C, веб-страница должна содержать только теги логического форматирования, а вид элементов должен задаваться через стили. Это позволяет распараллелить работу над дизайном и версткой сайта.

2) **Разное оформление для разных устройств.** С помощью стилей можно определить вид веб-страницы для разных устройств вывода: монитора, принтера, смартфона, планшета и

др. Также можно скрывать или показывать некоторые элементы документа при отображении на разных устройствах.

3) **Ускорение загрузки сайта.** При хранении стилей в отдельном файле, он кэшируется[1] и при повторном обращении к нему извлекается из кэша браузера. Такой подход позволяет существенно повысить скорость загрузки веб-страниц.

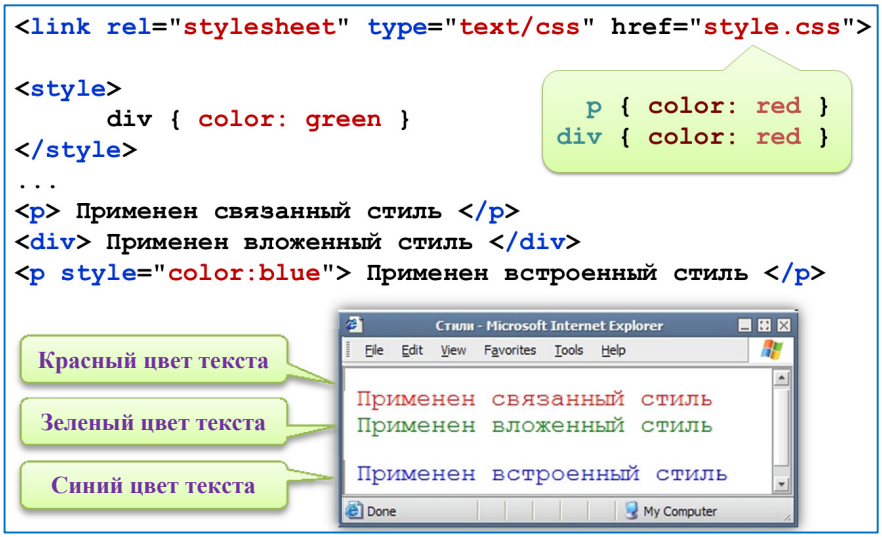


Рис. 2.11. Пример каскадирования стилей

4) **Единое стилевое оформление сайта.** Применение единообразного оформления заголовков, основного текста и других элементов создает преемственность между страницами сайта и облегчает пользователям его восприятие в целом. Также использование стилей существенно упрощает проектирование дизайна.

5) **Централизованное хранение.** Стили, как правило, хранятся в одном или нескольких специальных файлах, ссылка на ко-

[1] Кэш — участок памяти на локальном компьютере пользователя, куда браузер сохраняет файлы при первом обращении к сайту. При следующем обращении к сайту эти файлы уже не скачиваются по сети, а берутся с локального диска. За счет кэширования снижается время загрузки веб-страниц.

торые указывается во всех документах сайта. Благодаря этому удобно править стиль в одном месте, при этом оформление элементов автоматически меняется на всех страницах, которые связаны с указанным файлом. Вместо того чтобы модифицировать группу html-файлов, достаточно отредактировать один файл со стилем и оформление всей группы веб-страниц поменяется одновременно.

2.3. Виды селекторов CSS

Как говорилось ранее, каждый css-стиль имеет две основные части: селектор и блок объявления. Форматирующие свойства, которые размещены в блоке объявления, относятся лишь к оформлению. Ключевым фактором применения CSS к определенным элементам разметки HTML, является «маркер» определения любого стиля — **селектор**. Именно селектор определяет элементы, подлежащие форматированию. В примере кода селектор `h1` определяет, что форматированию указанными стилевыми правилами подлежат все заголовки первого уровня (элемент `h1`, задаваемый тегами `<h1>` и `</h1>`) веб-страницы:

```
h1 {
   fotn-size: 12pt;
   color: blue;
   background: tomato;
   border: 1px solid navy;
}
```

2.3.1. Селекторы типов

Селекторы типов (иногда называют *селекторами тегов*) являются весьма эффективным средством проектирования дизайна веб-страниц, поскольку определяют стиль всех экземпляров конкретного html-элемента.

Селекторы исключительно просто определить в CSS-стилях, так как они наследуют название форматируемых элементов — `p`, `h1`, `table`, `img` и т. д. (здесь и далее, согласно правилам общего обозначения синтаксических конструкций, в квадратных скобках размещаются необязательные фрагменты конструкции):

```
Тег {
   свойство1: значение [; свойство2: значение; ...]
}
```

Иными словами, в качестве селектора типов может выступать любой элемент HTML, для которого определяются правила форматирования, такие как: цвет, фон, размер и т. д. Например, html-код, представленный на рис. 2.12, имеет тег <p>, который в прописан без дополнительных атрибутов для его стилизации. Однако, вложенный единственный css-стиль с **селектором p**, определяет представление каждого абзаца страницы (на рис. 2.12 приведен один абзац, однако каждый объявляемый в данном контексте абзац будет иметь зеленый цвет текста и его выравнивание справа).

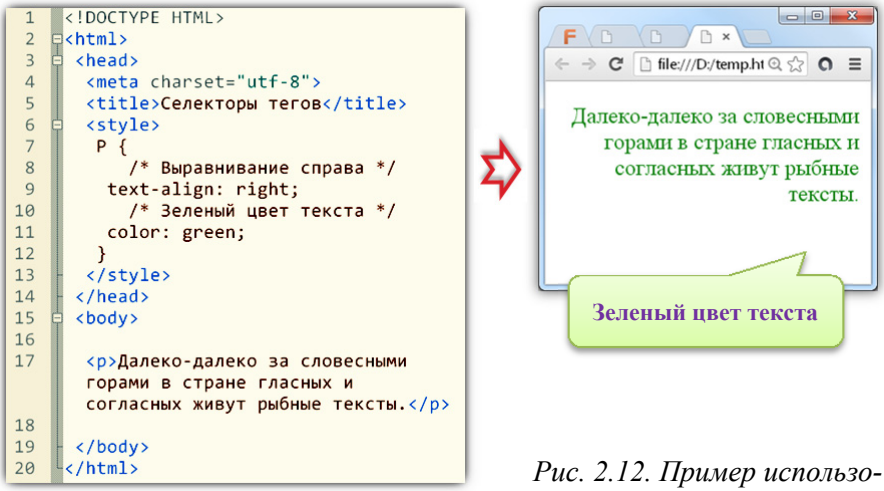


Рис. 2.12. Пример использования селектора тегов

Селекторы типов имеют свои недостатки. Если, например, необходимо сделать, чтобы не все абзацы веб-страницы выглядели одинаково, то простыми селекторами типов этого добиться не удастся.

2.3.2. Селекторы классов

Если, к примеру, нужно задать различия между элементом p, выделенным красным цветом и размером текста в 18 пунктов, и элементом p, со шрифтом по умолчанию, написанным черным цветом, то CSS предоставляет сразу несколько способов решения данной проблемы, самый простой из которых — **селекторы классов**.

Селекторы классов позволяют указать конкретный элемент веб-страницы, независимо от тегов:

```
Тег.имя_класса {
   свойство1: значение [; свойство2: значение; ...]
}
```

Например, используя селекторы классов легко можно создать селектор p.copyright и с его помощью выделить абзац p, содержащий информацию об авторских правах, не затрагивая остальные абзацы. Но как абзац, содержащий информацию об авторских правах, «привязать» к заданному для него классу?

Чтобы указать в коде HTML, что тег используется с определенным классом, к тегу добавляется атрибут class = “имя_класса”. То есть для примера с авторским правом, абзац p, содержащий информацию об авторских правах должен быть прописан в html-файле следующим образом:

```
<p class=“copyright”> … </p>
```

На рис. 2.13 приведен пример, показывающий преимущества использования селектора классов. Здесь продемонстрировано управление цветом текста. Для всех абзацев установлен зеленый

цвет текста, а для абзаца с классом “`cite`” — цвет текста переопределяется на цвет `navy` (оттенок синего).

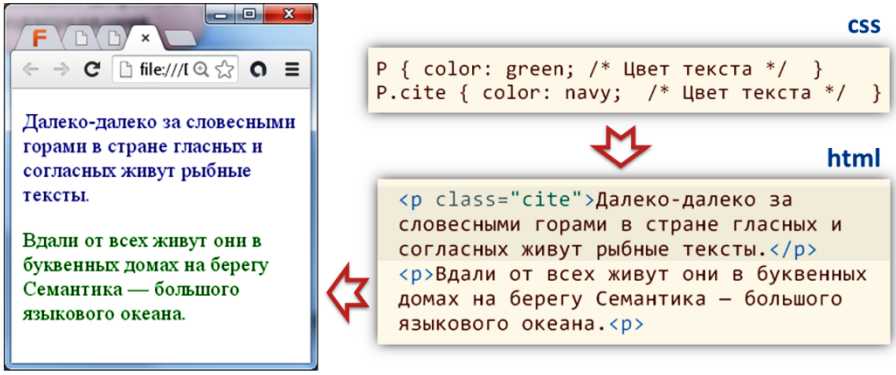


Рис. 2.13. Пример использования селектора классов (верхний абзац текста веб-страницы имеет цвет «navy», а нижний абзац — зеленый цвет)

Рассмотрим **правила**, которые необходимо иметь в виду, **именуя классы**:

1) При объявлении селектора классов, имя класса отделяется *точкой*: `.copyright`, `.cite` и пр. С ее помощью браузеры находят селекторы классов в таблице стилей CSS.

2) При именовании стилевых классов разрешается использование только *букв латинского алфавита*, *чисел*, *дефиса* и *знака подчеркивания*.

3) Название после точки всегда должно начинаться с символа — буквы латинского алфавита. Например, `.7dogs` — *неправильное* имя класса, а `.dogs7` — *правильное*. Можно называть классы, например, именами `.big-right` и `.main_image`, но не именами `.-bad` или `_as_bad`.

4) Имена стилевых классов чувствительны к регистру. Например, `.SIDEBAR` и `.sidebar` рассматриваются языком CSS как различные классы.

5) В атрибуте `class` начального тега элемента имя класса прописывается без точки. Например, класс с именем `.sidebar` можно задать элементу с тегом `<div class="sidebar">`.

Можно также использовать классы и без указания тега:

```
.имя_класса {
   свойство1: значение [; свойство2: значение; ...]
}
```

При такой записи класс можно применять к любому тегу. На рис. 2.14 продемонстрировано применение класса без указания тега. В стилях CSS объявлен селектор тега `p`, указывающий, что все абзацы должны иметь зеленый цвет текста. Также объявлен класс `.cite`, который определяет цвет текста цвета `navy`. В коде HTML показано, как можно переопределять цвета абзаца и отдельного слова, «подключая» класс `.cite` к отдельным элементам `p` и `strong`.

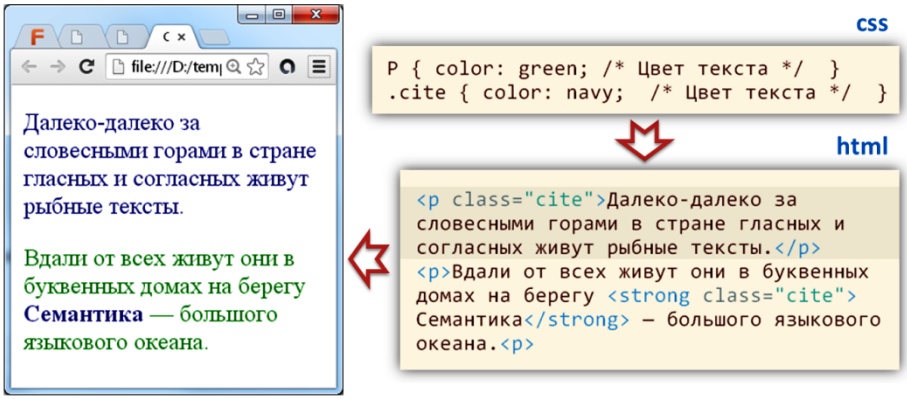


Рис. 2.14. Пример использования селектора классов без указания имени тега (верхний абзац текста веб-страницы имеет цвет «navy», а нижний абзац — зеленый цвет, однако слово «Семантика» получило выделение цветом «navy»)

К любому тегу одновременно можно **добавить несколько классов**, перечисляя их в атрибуте `class` через пробел. Например, запись

```
<p class="copyright cite"> … </p>
```

означает, что для элемента `p` будут применяться стилевые свойства вначале класса `.copyright`, а затем `.cite` (т. е. в случае применения одних и тех же стилевых правил с разными значениями, окончательные значения будут из класса `.cite`).

Классы удобно использовать, когда нужно применить стиль к разным элементам веб-страницы (рис. 2.15): ячейкам таблицы, ссылкам, абзацам и др.

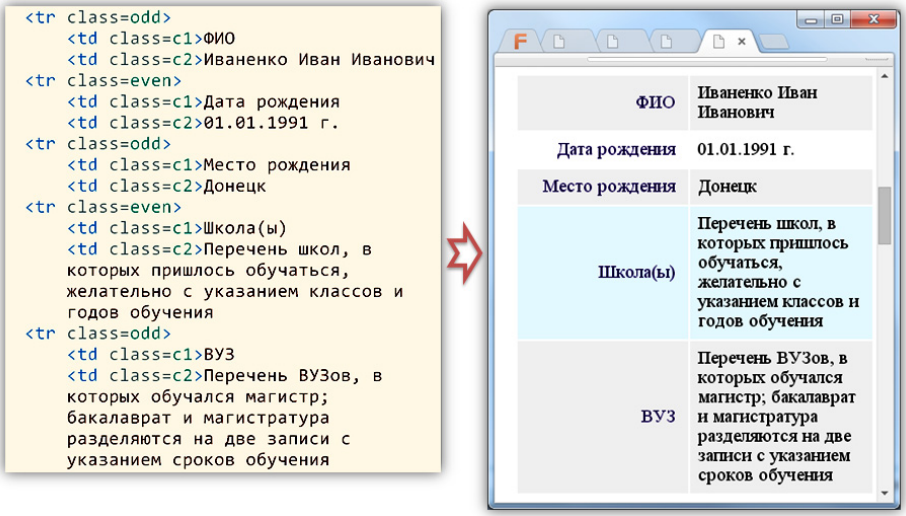


Рис. 2.15. Применение классов на примере оформления таблицы на странице «Резюме» в шаблоне персонального сайта магистров

2.3.3. ID-селекторы

В языке CSS ID-селектор предназначен для идентификации уникальных частей веб-страниц, таких как заголовок (хедер), панель навигации, основная информационная область содержимого, нижняя часть (подвал, футер) и т. д. Как и при использовании селектора класса, создается **идентификатор** (ID), и затем применяется к html-коду веб-страницы.

А в чем же различие между классами и идентификаторами? Дело в том, что ID-селекторы имеют дополнительное специфическое применение. Например, в веб-страницах с использованием кода JavaScript или в очень объемных страницах. Так, существует несколько вынужденных причин для применения ID-селекторов. В общем случае, идентификатор (ID-селектор) определяет **уникальное имя элемента**, которое используется для изменения его стиля и обращения к нему через скрипты. В отличие от классов, идентификаторы должны быть уникальны, т. е. встречаться в коде документа только один раз. Идентификатор объявляется следующим образом:

```
#имя_идентификатора {
   свойство1: значение [; свойство2: значение; ...]
}
```

Применение идентификаторов в HTML схоже с использованием классов, но требует другого атрибута с соответствующим названием — `id`. На рис. 2.16 показано объявление идентификатора `#block` и применению его свойств к html-элементу `div` с помощью атрибута `id = “block”`.

Точно так же, как и в примере при объявлении классов, для указания, что последний абзац веб-страницы — единственный с информацией об авторских правах, можно создать ID-стиль под названием `#copyright` и применить его к тегу этого абзаца:

```
<p id=“copyright”> … </p>
```

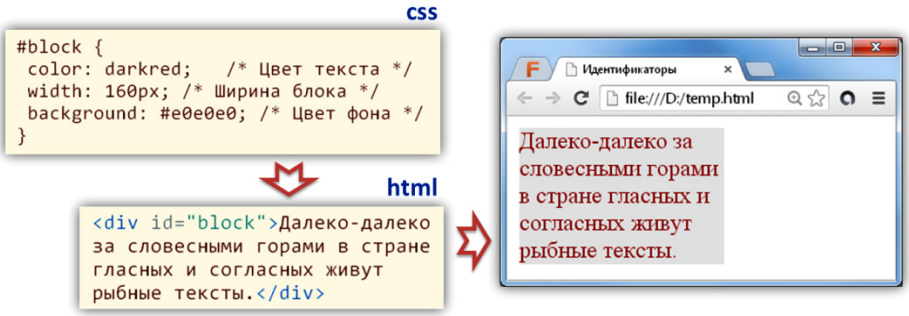


Рис. 2.16. Пример использования ID-селектора (загруженная в браузер веб-страница содержит текстовый блок шириной в 160 пикселей с темно-красным (darkred) цветом текста и серым (#E0E0E0) цветом фона)

Как и при использовании классов, **идентификаторы можно применять к конкретному тегу**:

```
Тег#имя_идентификатора {
   свойство1: значение [; свойство2: значение; ...]
}
```

На рис. 2.17 показано, что селектор `p#block` назначит элементу объявленные в нем стилевые правила, только если в html-разметке встретится элемент `p` с атрибутом `id = “block”`.

2.3.4. Стилизация групп тегов

Иногда необходимо быстро применить одинаковое форматирование сразу к нескольким различным элементам веб-страницы. Например, нужно, чтобы все заголовки разных уровней имели один и тот же цвет текста — `#FAC773`. Создание отдельного стиля для каждого заголовка определенного уровня — слишком объемная работа. А если потом возникнет необходимость изменить цвет всех заголовков одновременно, то придется все обновлять.

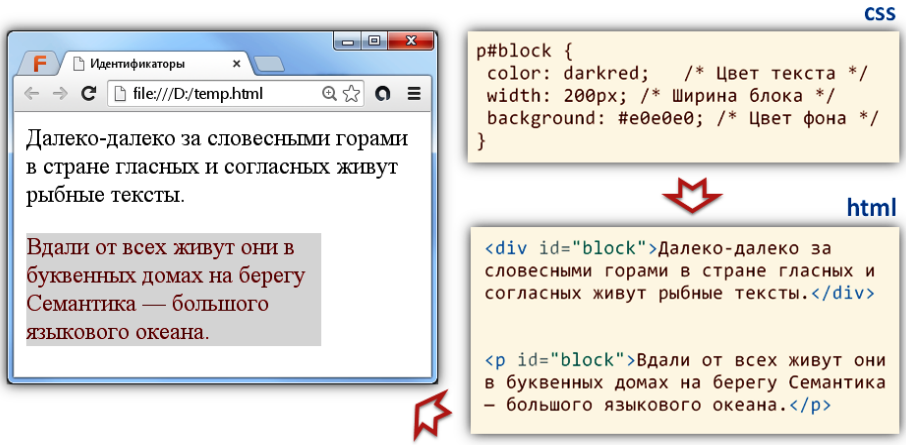


Рис. 2.17. Пример использования ID-селектора к конкретному тегу (идентификатор «block» для элемента «div» не сработал, так как он был объявлен только для элемента «p»)

Для решения этой задачи лучше использовать **групповые селекторы**, которые позволяют применить стиль, описав его лишь один раз, одновременно ко всем элементам [9].

Для работы с группой селекторов нужно создать список, в котором **один селектор отделен от другого запятыми**. Таким образом, получаем:

```
h1, h2, h3, h4, h5, h6 { color: #FAC773; }
```

Здесь приведены только селекторы типов, но можно использовать любые виды (или их сочетание). Например, стиль с групповым селектором, который определяет одинаковый цвет шрифта для `<h2>`, `<ol>`, любых других, принадлежащих классу `.copy`, а также тегу `<p>` с идентификатором `#banner`, записывается следующим образом:

```
h2, ol, .copy, p#banner { color: #FAC773; }
```

Групповые селекторы можно рассматривать как подручное средство для применения одинаковых свойств различных элементов. CSS предоставляет **универсальный селектор** * для выборки всех тегов веб-страницы. Например, если нужно, чтобы все отображалось *полужирным шрифтом*, то необходимо перечислить все используемые на веб-странице элементы:

```
a, p, img, h1, h2, h3, h4, ... { font-weight: bold; }
```

В этом случае использование символа * — более быстрый способ сообщить CSS о выборке всех html-элементов веб-страницы:

```
* { font-weight: bold; }
```

2.3.5. Контекстные селекторы

Контекстный селектор (**селектор потомков**) состоит из простых селекторов разделенных пробелом

```
Тег1 Тег2 {
   свойство1: значение [; свойство2: значение; ...]
}
```

В этом случае стиль будет применяться к **Тегу2** если он — **потомок Тега1**, т. е. размещается внутри Тега1.

На рис. 2.18 показан пример использования контекстных селекторов. Конструкция контекстных селекторов `p em` устанавливает цвет текста красным для элемента `em`, если он является вложенным в элемент `p` с любым уровнем вложенности.

Более широкие возможности контекстные селекторы дают при использовании идентификаторов и классов. Это позволяет устанавливать стили только для элемента с определенным идентификатором, который располагается внутри элемента определенного класса.

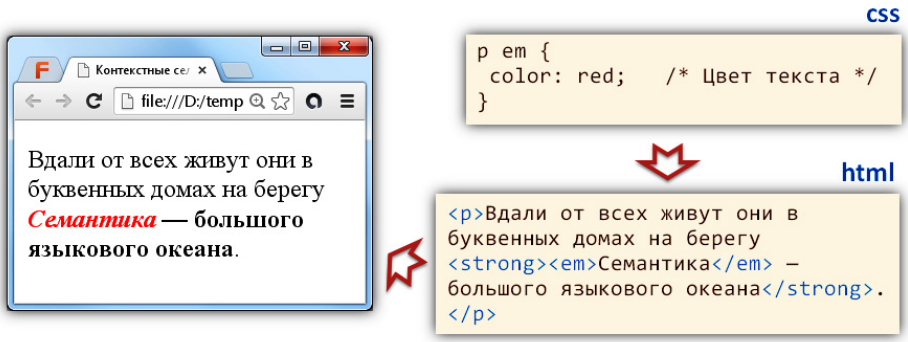


Рис. 2.18. Пример использования контекстных селекторов (слово «Семантика» написано красным цветом)

2.3.6. Дочерние селекторы

Дочерним называется элемент, который непосредственно располагается внутри родительского элемента (т. е. находится на первом уровне вложенности).

Подобно контекстных селекторов в CSS возможно форматировать вложенные элементы других тегов с помощью **селектора дочерних элементов**, который использует дополнительный символ — угловую скобку «>» — для указания отношения между двумя элементами:

```
Тег1 > Тег2 {
   свойство1: значение [; свойство2: значение; ...]
}
```

Например, `body > ul` выбирает любой элемент `ul`, дочерний по отношению к `body`.

В отличие от контекстного селектора, который применяется ко всем потомкам (т. е. всем вложенным элементам), селектор дочерних элементов позволяет определить конкретные дочерний и родительский элементы. На рис. 2.19 показан пример, в котором к элементу `em` применится цвет текста красным цветом, только в том случае, если `em` является дочерним по отношению к

элементу p. В приведенном коде HTML размещены два элемента em. Поскольку у em различные родительские элементы, у первого это элемент p, а у второго — элемент strong. Значит стили, заданные дочерним селектором применяются к первому элементу em, что подтверждает левая часть рис. 2.19.

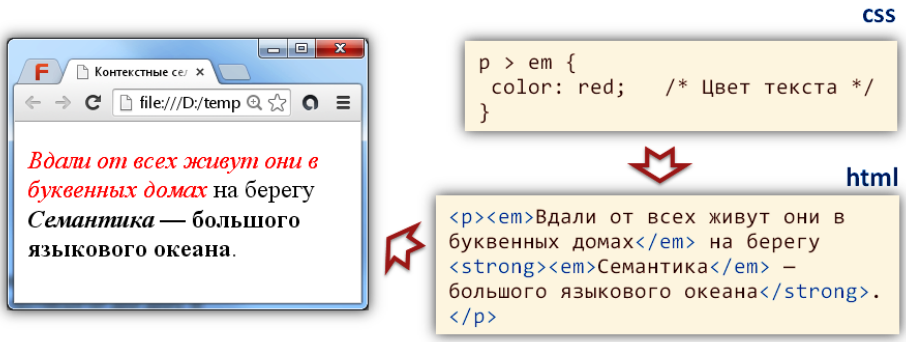


Рис. 2.19. Пример использования дочерних селекторов (фраза «Вдали от всех живут они в буквенных домах» выделена красным цветом)

Итак, дочерние селекторы по своей логике похожи на контекстные селекторы. Отличия:

- стиль к дочернему селектору применяется только в том случае, когда он является прямым потомком, иными словами, непосредственно располагается внутри родительского элемента;
- стиль к контекстному селектору применяется, когда он является потомком с любым уровнем вложенности.

2.3.7. Соседние селекторы

Соседними (**смежными**) называются такие элементы HTML, которые следуют непосредственно друг за другом в коде html-документа (см. п. 1.3.1). Для управления стилем соседних элементов используется символ плюса «+», который устанавливается между двумя селекторами.

Общий синтаксис **соседних селекторов**:

```
Тег1 + Тег2 {
   свойство1: значение [; свойство2: значение; ...]
}
```

В этом случае стиль будет применяться к **Тегу2** только в том случае, если он является **смежным** для **Тега1** и следует сразу после него.

В примере на рис. 2.20 только в одном случае элемент `em` следует за элементом `strong`, именно для этого случая и применяется стиль `color: red`. То, что `strong` и `em` расположены внутри контейнера `<p>`, никак не влияет на их отношение.

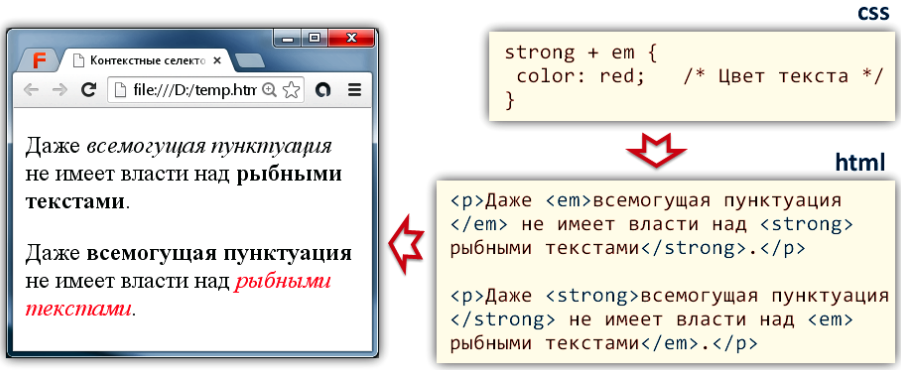


Рис. 2.20. Пример использования соседних селекторов (фраза «рыбными текстами» выделена красным цветом)

Соседние селекторы удобно использовать для тех тегов, к которым автоматически добавляются отступы, чтобы самостоятельно регулировать величину отбивки. Например, если подряд идут теги `<h1>` и `<h2>`, то расстояние между ними легко регулировать как раз с помощью соседних селекторов. Аналогично дело обстоит и для идущих подряд тегов `<h2>` и `<p>`, а также в других подобных случаях [10].

2.3.8. Селекторы атрибутов

Каскадные таблицы стилей обеспечивают возможность форматирования тегов на основе выборки любых содержащихся в них атрибутов. Иными словами, **селекторы атрибутов** позволяют установить стиль по присутствию определенного атрибута элемента или его значения

Простой селектор атрибута устанавливает стиль, если задан специфичный атрибут:

```
[атрибут] { Описание правил стилей ... }
Селектор[атрибут] { Описание правил стилей ... }
```

Здесь стиль применяется к тем тегам, внутри которых добавлен указанный атрибут. Если указан конкретный селектор, то между названием селектора и квадратной скобкой пробел не допускается. Атрибут со значением устанавливает стиль, если задано определенное значение специфичного атрибута:

Селектор атрибута со значением устанавливает стиль, если задано определенное значение специфичного атрибута:

```
[атрибут="значение"] { ... }
Селектор[атрибут="значение"] { ... }
```

Для показанной в первой строке синтаксической записи, стиль применяется ко всем html-элементам, которые содержат указанное значение. А для второй строки — только к определенным селекторам.

Допускается записывать значение атрибута без кавычек, если значение содержит латинские буквы и не содержит символов пробела.

Значение атрибута начинается с определенного текста. Устанавливает стиль для элемента, если значение атрибута тега *начинается с указанного текста* (конструкция «^=»):

```
[атрибут^="значение"] { ... }
Селектор[атрибут^="значение"] { ... }
```

Значение атрибута оканчивается определенным текстом. Устанавливает стиль для элемента в том случае, если значение атрибута *оканчивается указанным текстом* (используется конструкция «$=»):

```
[атрибут$="значение"] { ... }
Селектор[атрибут$="значение"] { ... }
```

Значение атрибута содержит указанный текст. Возможны варианты, когда стиль следует применить к тегу с определенным атрибутом, при этом частью его значения является некоторый текст. При этом точно не известно, в каком месте значения включен данный текст — в начале, середине или конце. В этом случае можно использовать следующий синтаксис (используется конструкция «*=»):

```
[атрибут*="значение"] { ... }
Селектор[атрибут*="значение"] { ... }
```

Одно из нескольких значений атрибута. Некоторые значения атрибутов могут перечисляться через пробел, например имена классов. Чтобы задать стиль при наличии в списке требуемого значения применяются следующие записи (используется конструкция «~=»):

```
[атрибут~="значение"] { ... }
Селектор[атрибут~="значение"] { ... }
```

Стиль применяется в том случае, если у атрибута имеется указанное значение или оно входит в список значений, разделяемых пробелом.

Дефис в значении атрибута. В именах идентификаторов и классов разрешено использовать *символ дефиса* «-», что позволяет создавать интуитивно понятные значения для атрибутов `id` и `class`. Для изменения стиля элементов, в значении которых применяется дефис, можно воспользоваться следующим синтаксисом (используется конструкция «|=»):

```
[атрибут|="значение"] { ... }
Селектор[атрибут|="значение"] { ... }
```

Комбинирование атрибутов. Все вышеперечисленные методы можно комбинировать между собой, определяя стиль для элементов, которые содержат два и более атрибута. В подобных случаях квадратные скобки идут друг за другом без пробела:

```
[атрибут1="значение1"][атрибут2="значение2"] { ... }
Тег[атрибут1="значение1"][атрибут2="значение2"] { ... }
```

Примеры применения селекторов атрибутов:

1. Применение стиля к ссылкам, в атрибуте `href` которых встречается слово "`mysite`":

```
[href*="meysite"] {
     background: yellow;      /* желтый цвет фона */
}
```

2. Применение стиля (текст зеленого цвета) к заголовку `h3`, если одно из имен класса у элемента-предка задано как "`block`":

```
[class~="block"] h3 { color: green }
```

3. Применение стиля к **внешним ссылкам** (ссылкам с абсолютной адресацией):

```
a[href^="http://"], a[href^="https://"] {
    font-weight: bold;    /* полужирное начертание*/
}
```

Для удобства использования, составим таблицу, в которую сведем особенности и примеры задания селекторов атрибутов [11] (табл. 2.1).

Таблица 2.1. Обзор селекторов атрибутов

Пример	Название	Описание
`a[target]`	Селектор наличия атрибута	Выбирает элемент, если данный атрибут присутствует
`a[href= "http://tom.com/"]`	Селектор атрибута =	Выбирает элемент, если значение данного атрибута в точности соответствует указанному
`a[href*="login"]`	Селектор атрибута *=	Выбирает элемент, если значение данного атрибута содержит, по крайней мере, один экземпляр указанного текста
`a[href^="https:// "]`	Селектор атрибута ^=	Выбирает элемент, если значение данного атрибута начинается с указанного текста
`a[href$=".pdf"]`	Селектор атрибута $=	Выбирает элемент, если значение данного атрибута заканчивается указанным текстом
`a[rel~="tag"]`	Селектор атрибута ~=	Выбирает элемент, если значение данного атрибута разделено пробелами и точно совпадает с одним указанным словом
`a[lang|="en"]`	Селектор атрибута |=	Выбирает элемент, если значение данного атрибута разделено дефисом и начинается с указанного слова

2.3.9. Псевдоклассы

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя. Примером такого состояния служит текстовая ссылка, которая меняет свой цвет при наведении на нее курсора мыши.

При использовании псевдоклассов браузер не перегружает текущий документ, поэтому с помощью псевдоклассов можно получить разные динамические эффекты на странице. Синтаксис объявления селектора с псевдоклассом:

```
Селектор:псевдокласс { Описание правил стилей ... }
```

Условно все псевдоклассы можно разделить на три группы:

1. Определяющие состояние элементов.
2. Имеющие отношение к дереву элементов (например, псевдокласс `:first-child` — применяется к первому дочернему элементу селектора).
4. Указывающие язык текста (например, псевдокласс `:lang` определяет язык, который используется в документе или его фрагменте).

Псевдоклассы, определяющие состояния элементов, наиболее часто используются для установки разных стилей ссылок:

`:link` — ссылки на еще не посещенные страницы, например,

```
a:link { color: blue; }
```

`:visited` — ссылки на уже посещенные страницы, например,

```
a:visited { color: #660099; }
```

`:active` — используется для активных ссылок:

```
a:active { background-color: #FFFF00; }
```

:hover — ссылка, над которой находится указатель мыши:

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

Псевдокласс :hover можно добавлять и к другим элементам документа, например, чтобы в таблице строки меняли свой цвет при наведении на них курсора мыши.

Особенности и назначения псевдоклассов других типов предлагаются студентам для самостоятельного изучения. Ниже перечислены некоторые псевдоклассы, согласно их функциональным особенностям [11]:

:first-child выбирает элемент, если это первый «ребенок» в родительском элементе.

:last-child выбирает элемент, если это последний «ребенок» в родительском элементе

:only-child выбирает элемент, если он является единственным элементом в родителе

Например, селектор li:first-child определяет первый пункт списка, в то время как li:last-child определяет последний пункт списка, таким образом, выбираются строки 2 и 10. Селектор div:only-child ищет , который является единственным ребенком в родительском элементе, без каких-либо других родственных элементов. В этом случае выбирается строка 4, т. к. это единственный в данном пункте списка:

CSS

```
1  li:first-child {...}
2  li:last-child {...}
3  div:only-child {...}
```

HTML

```
1  <ul>
2    <li>Этот пункт будет выбран</li>
3    <li>
4      <div>Этот div будет выбран</div>
5    </li>
6    <li>
```

HTML

```
      <div>...</div>
      <div>...</div>
    </li>
    <li>Этот пункт будет выбран</li>
  </ul>
```

`:first-of-type` выбирает первый элемент своего типа внутри родителя.

`:last-of-type` выбирает последний элемент этого типа внутри родителя.

`:only-of-type` выбирает элемент, если он является единственным такого типа в родителе.

Например, псевдоклассы `p:first-of-type` и `p:last-of-type` выберут, соответственно, первый и последний абзацы в статье, независимо от того, являются ли они на самом деле первыми или последними детьми в статье. Строки 3 и 6 выбраны этими селекторами. Селектор `img:only-of-type` определяет изображение в строке 5, как единственное изображение появляющееся в документе:

CSS

```
p:first-of-type {...}
p:last-of-type {...}
img:only-of-type {...}
```

HTML

```
<article>
  <h1>...</h1>
  <p>Этот абзац будет выбран</p>
  <p>...</p>
  <img src="#">  <!-- Это изображение будет выбрано -->
  <p>Этот абзац будет выбран</p>
  <h6>...</h6>
</article>
```

Также есть несколько псевдоклассов, которые выбирают элементы, основанные на номере или алгебраическом выражении: Эти псевдоклассы включают в себя **:nth-child(n)**, **:nth-last-child(n)**, **:nth-of-type(n)** и **:nth-last-of-type(n)**. Все эти уникальные псевдоклассы начинаются с nth и принимают число или выражение внутри круглых скобок, которое обозначается символом n. Это число или выражение, которое точно определяет, какой элемент или элементы, должны быть выбраны.

Последние, рассматриваемые в этом издании псевдоклассы, это **:empty** и **:not**.

Псевдокласс :empty позволяет выбрать элементы, которые не содержат дочерние элементы или текст (комментарии и пустой текст не считаются дочерними элементами).

На примере ниже использование селектора div:empty определит без дочерних элементов или текста. Поэтому при обработке браузером html-файла будут выбраны в строках 3 и 4, поскольку они совершенно пустые. Даже несмотря на то, что второй содержит комментарий, он не считается вложенным текстом, поэтому этот остается пустым. Первый содержит текст, третий содержит один пробел, а последний содержит дочерний элемент , таким образом, они все исключены и не выбраны:

CSS

```
div:empty {...}
```

HTML

```
1  <div>Работаем с псевдоклассом empty.</div>
2                    <!-- Следующий div будет выбран -->
3  <div><!-- Текста пока нет --></div>
4  <div></div>       <!-- Этот div будет выбран -->
5  <div>  </div>
6  <div><strong></strong></div>
```

Псевдокласс :not(x) принимает аргумент и отфильтровывает выборку, которая будет сделана. Например, селектор p:not(.int-base) использует псевдокласс :not для определе-

ния каждого абзаца без класса `int-base`. Элемент абзаца определяется в начале селектора, затем следует псевдокласс `:not(x)`. Внутри скобок идет селектор отрицания, в данном случае класс `int-base`.

В примере ниже оба селектора `div:not(.aws)` и `:not(div)` используют псевдокласс `:not(x)`. Селектор `div:not(.aws)` определяет любой `<div>` без класса `aws`, в то время как селектор `:not(div)` определяет элемент, который не является `<div>`. В результате выбирается `<div>` в строке 1, а также два раздела в строках 3 и 4. В строке 2 — единственный не выбранный элемент это `<div>` с классом `aws`, так как он выходит за пределы двух псевдоклассов.

CSS

```
1  div:not(.aws) {...}
2  :not(div) {...}
```

HTML

```
1  <div>Этот элемент будет выбран</div>
2  <div class="aws">...</div>
3  <section>Этот раздел будет выбран</section>
4  <section class="aws">
5     Этот раздел будет выбран
6  </section>
```

2.4. Единицы измерения, способы задания цветов, шрифтовое оформление

2.4.1. Единицы измерения в CSS

Для задания размеров различных элементов, в CSS используются **абсолютные** и **относительные единицы** измерения. Абсолютные единицы не зависят от устройства вывода, а относительные единицы определяют размер элемента относительно значения другого размера.

Начнем, пожалуй, с основной единицы измерения — **пикселя** (px), которую можно условно отнести к каждой группе единиц [12].

Количество пикселей задается в настройках разрешения экрана, один px — это именно один пиксель на экране. Поэтому, отображая на экране свое содержимое, браузер в итоге **пересчитает все единицы измерения в пиксели**.

Размер в пикселях можно задавать дробными числами, например, 15.33px. Это приемлемо еще и потому, что браузер сам использует дробные значения пикселей для внутренних вычислений. Однако при окончательном отображении дробные значения пикселей округляются и становятся целыми.

Абсолютные единицы применяются реже, чем относительные и, как правило, при работе с текстом (рис. 2.21):

- ✓ in — **дюйм** (1 дюйм равен 2,54 см).
- ✓ cm — **сантиметр**.
- ✓ mm — **миллиметр** (1 миллиметр равен 1/10 сантиметра**)**.
- ✓ pt — типографский **пункт** (1 пункт равен 1/72 дюйма). В основном используется для указания размера шрифта. Повсеместное использование пунктов в компьютерных текстах возникло благодаря текстовым редакторам и издательским системам.
- ✓ pc — типографская **пика** (1 пика равна 12 пунктам).

Недостаток использования на практике абсолютных единиц изменения в том, что браузер в итоге пересчитывает все эти значения в пиксели[1]: 1mm = 3.8px; 1cm = 38px; 1pt = 4/3px; 1pc = 16px.

Поэтому особого смысла в их употреблении нет.

---

[1] Здесь уместно ответить на следующий вопрос. А как так получается, что если пиксель может быть разным, в зависимости от экрана, а, например сантиметр — эталон длины, одна сотая метра, — то разве 1 см всегда будет равен 38 пикселей? Дело в том, что при расчетах под пикселем понимается «сферический пиксель в вакууме», т. е. точка на «стандартизованном экране», характеристики которого описаны в спецификации CSS [13]. Поэтому ни о каком соответствии css-величины cm к реальному сантиметру нет — это полностью производная единица измерения (условный сантиметр).

```
<html>
 <head>
  <meta http-equiv="Content-Type"
  content="text/html; charset=utf-8">
  <title>Абсолютные единицы</title>
  <style type="text/css">
   .in { font-size: 0.5in; }
   .mm { font-size: 8mm; }
   .pt { font-size: 24pt; }
  </style>
 </head>
 <body>
  <p class="in">Размер 0.5 дюйма</p>
  <p class="mm">Размер 8 миллиметров</p>
  <p class="pt">Размер 24 пункта</p>
 </body>
</html>
```

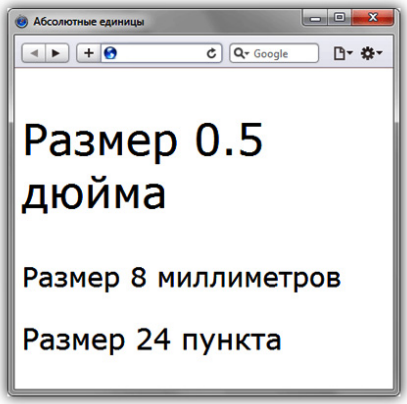


Рис. 2.21. Использование абсолютных единиц измерения

Относительные единицы обычно используют для работы с текстом, либо когда требуется вычислить процентное соотношение между элементами (рис.2.22):

✓ `em` — **высота шрифта текущего элемента**. Масштабируемая единица «em» равна текущему `font-size`, например, если `font-size = 12pt`, то `1em` равен `12pt`.

В спецификации CSS [13] указаны также единицы `ex` и `ch`, которые означают размер символа «x» и размер символа «0». Эти единицы используются чрезвычайно редко, так как «размера шрифта» `em` обычно вполне достаточно.

✓ % — **процент** от значения свойства родителя с тем же названием.

✓ `rem` — аналог `em`, однако **задает размер относительно размера шрифта элемента** `<html>`. С помощью этой единицы измерения можно получить удобное масштабирование для шрифтов, не влияющее на другие элементы.

Во всех современных браузерах поддерживаются новые единицы измерения из стандарта CSS Values and Units 3 [14]:

✓ `vw` — 1% **ширины** окна.

✓ `vh` — 1% **высоты** окна.

✓ `vmin` — **наименьшее** из (vw, vh).
✓ `vmax` — **наибольшее** из (vw, vh).

Эти значения были созданы, в первую очередь, для поддержки мобильных устройств. Их основное преимущество — любые размеры, которые в них заданы, автоматически масштабируются при изменении размеров окна.

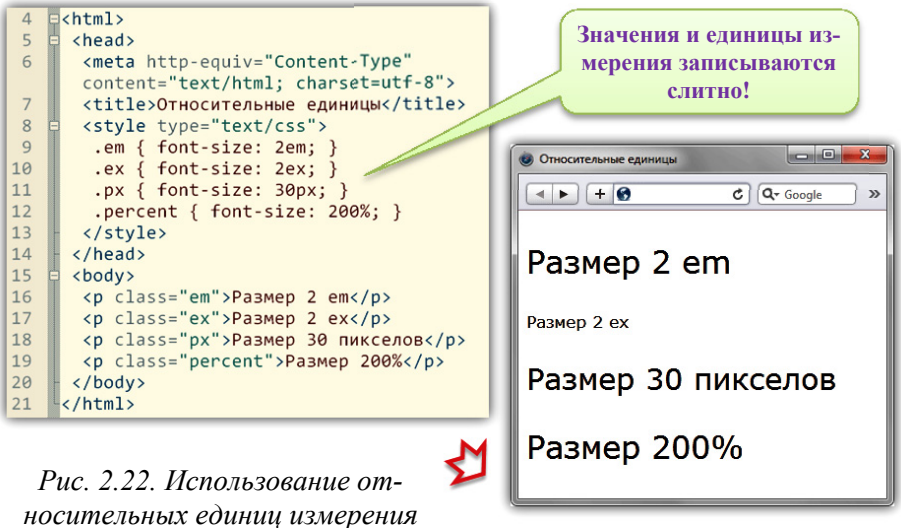


Рис. 2.22. Использование относительных единиц измерения

2.4.2. Способы задания цветов в CSS

Цвет в CSS задается одним из двух путей: с помощью шестнадцатеричного кода и по названию некоторых цветов. Преимущественно используется способ, основанный на шестнадцатеричной системе исчисления, как наиболее универсальный.

Спецификация CSS предоставляет **четыре удобных способа задать цвет**:

1. По имени. Используются англоязычные названия цветов, например «`blue`» (черный) или «`red`» (красный).

Цвета, заданные по имени (именованные цвета), как правило, собраны в соответствующие таблицы, в которых каждому имени

соответствуют образец (пробник) цвета и его код в шестнадцатеричном и десятичном форматах. Таких таблиц в Интернете огромное множество, среди которых любой, даже самый искушенный, пользователь найдет для себя исчерпывающую информацию. На рис. 2.23 показан вариант таблицы (палитры) цветов HTML и CSS.

Рис. 2.23. Палитра цветов HTML и CSS

2. В шестнадцатеричном формате (Base-16, hex) цветовой модели RGB [15, 16] — в формате `#rr`$_{16}$`gg`$_{16}$`bb`$_{16}$ (например, `#71C354`), где решетка «#» — признак задания значений цветовых каналов в формате Base-16, `rr`$_{16}$ — насыщенность красным цветом (двузначное шестнадцатеричное число), `gg`$_{16}$ — зеленым, а `bb`$_{16}$ — синим.

Например, белый цвет задается кодом `#FFFFFF`, а черный — кодом `#000000`. Формат `#rgb` — сокращенная запись hex-формата задания цвета, которая приводит к повторению значе-

ния шестнадцатеричной цифры в каждом цветовом компоненте. Например, код `#363` эквивалентен записи `#336633`.

Рассмотрим шестнадцатеричные представления цветов на примере следующего кода цвета:

Как уже было отмечено, символ решетки «#» перед числом означает, что оно шестнадцатеричное. Первые две цифры (`FA`) определяют **красную составляющую цвета**, цифры с третьей по четвертую (`8E`) — **зеленую**, а последние две цифры (`47`) — **синюю** (коды цвета в CSS не чувствительны к регистру задаваемых их символов):

Приведем условные *правила, чтобы легче было ориентироваться в шестнадцатеричных цветах*:

1) если значения компонент цвета одинаковы (например: `#D6D6D6`), то получится серый оттенок;
2) чем больше число, тем светлее цвет: цвета от `#000000` (черный) до `#FFFFFF` (белый);
3) цвет со значением `#FF0000` самый красный из возможных красных оттенков; аналогично — с зеленым цветом (`#00FF00`) и синим цветом (`#0000FF`);
4) желтый цвет (`#FFFF00`) получается смешением красного с зеленым, голубой цвет (`#00FFFF`) получается за счет объединения синего и зеленого цвета и т. д.

Цвета по шестнадцатеричным значениям *не обязательно подбирать эмпирическим путем*. Для этой цели подойдет графический редактор (например, Adobe Photoshop), программы для работы с цветом (например, ColorMania), или специализированные веб-сервисы (например, Color Scheme).

3. В десятичном формате модели RGB — `rgb(`rr_{10}`, `gg_{10}`, `bb_{10}`)`, где значение каждой компоненты цвета rr_{10}, gg_{10} и bb_{10} задаются абсолютными значениями и могут ва-

рьироваться от 0 до 255. В таком формате белый цвет задается кодом `rgb(255,255,255)`, красный — кодом `rgb(255,0,0)`, синий — кодом `rgb(0,255,0)`, зеленый — кодом `rgb(0,0,255)`, а черный — кодом `rgb(0,0,0)`.

4. В формате модели RGB с указанием процентного содержания цветовых компонент — `rgb(rr%, gg%, bb%)`, где значение каждой компоненты цвета может варьироваться относительными значениями от 0% до 100%. В таком случае белый цвет задается кодом `rgb(100%,100%,100%)`, а нейтральный серый — кодом `rgb(50%,50%,50%)`.

***

Также для определения кода цвета по образцу, просмотра цвета по введенному коду, получения параметров цветов в различных цветовых моделях и многих других, связанных с цветом, операций предназначен целый ряд специализированных программ и веб-сервисов, самые популярные из которых представлены в книге «Веб-типографика. Искусство оформления текстов для Интернета» [15].

В виду особенностей отображения цветов, один и тот же цвет в веб-странице может по-разному отображаться на компьютерах пользователей (в смысле, на устройствах вывода пользователей — мониторах, экранных (телевизионных) панелей, проекторах, экранах мобильных устройств и пр.). Кроме того, браузеры различных компаний интерпретируют один и тот же цвет поразному. Когда браузер не может точно передать цвет, он начинает подбирать похожий, или начинает смешивать несколько «соседних» цветов, что в итоге может привести к полному искажению цветовой гаммы сайта.

Цвета, которые будут гарантированно точно отображаться во всех браузерах, называют **безопасными веб-цветами** (англ. *browser-safe colors*). Безопасные цвета были выведены математически. Безопасные цвета были выведены математически. Так, для получения безопасного цвета в значениях красной, зеленой и синей компоненты нужно использовать только десятичные значения `0`, `51`, `102`, `153`, `204`, `255` (и никакие другие). Шестна-

дцатеричная запись цвета должна состоять только из 00, 33, 66, 99, CC, FF. Например, #FFCC00 и rgb(204,102,51) — коды безопасных веб-цветов. Однако, учитывая специфичность интерпретации цвета разными операционными системами и мониторами разных производителей, можно справедливо оговориться и уточнить, что абсолютно безопасного веб-цвета не существует!

Поэтому, на сегодняшний день «безопасная» палитра не обеспечивает цветового постоянства. Она, скорее всего, ограничивает веб-дизайнеров, которые желают выйти за границы предложенного набора безопасных веб-цветов.

В CSS для задания цвета элемента используется свойство color. Ниже приведен пример задания синего цвета для текста параграфа всеми указанными выше вариантами:

CSS

```
p { color: blue; }
p { color: rgb(0,0,255); }
p { color: #00F; }                  /* # r  g  b  */
p { color: #0000ff; }               /* # rr gg bb */
p { color: rgb(0%, 0%, 100%);
```

Для задания цвета фона элемента в CSS используется свойство background-color, которое может использоваться как самостоятельное свойство, а может быть включено в сокращенное свойство background — краткий вариант записи для свойств background-color, background-image, background-repeat, background-attachment и background-position.

Цветовая модель RGB расширена в спецификации CSS3 так, чтобы включать значение «альфа-канала», которое отвечает за величину прозрачности цвета. Полученная цветовая модель имеет аббревиатуру RGBA (буква «A» в аббревиатуре от «Alpha» (альфа-канал)). С помощью данной модели можно установить не только необходимое сочетание красного, зеленого и синего, но также определить прозрачность указанного цвета.

CSS

```
h1 { color: rgba(0, 255, 0, 0.5); }
```

Так же, как в `rgb(…)`, первые три значения отвечают за сочетание красного, зеленого и синего цветов. Они могут принимать как целочисленные значения в диапазоне 0÷255, так и процентные соотношения в промежутке от 0 до 100 %. Четвертое значение определяет степень прозрачности в диапазоне от 0 (абсолютно прозрачный) до 1 (совершенно непрозрачный).

В приведенном выше примере для отображения текста заголовков, заключенного в тегах `<h1>…</h1>`, будет использоваться зеленый цвет с прозрачностью 50 %. В отличие от значений в формате RGB, значение цвета в формате RGBA не имеет шестнадцатеричной записи.

Все современные браузеры последних версий поддерживают RGBA-цвета. Однако для адаптации кода CSS под абсолютно все версии браузеров достаточно первым объявлением указать цвет в формате RGB, а вторым — задать цветовой оттенок в формате RGBA — для браузеров, поддерживающих данную цветовую модель. Например:

CSS

```
h1 {
        /* темно-серый цвет (gray) */
color: rgb(50%, 50%, 50%);
        /* черный с 50 % прозрачностью */
color: rgba(0, 0, 0, 0.5);
}
```

В данном примере строка «`color: rgba(0,0,0,0.5);`» будет проигнорирована не поддерживающим данную спецификацию браузером, и текст заголовка `h1` будет отображен непрозрачным серым цветом (выполнение строки «`rgb(50%,50%,50%)`»).

В остальных браузерах текст заголовка `h1` отображается черным цветом с прозрачностью 50% (выполнение последней строки «`color: rgba(0,0,0,0.5)`»). Кроме задания цвета элемента (в частности, цвета текста) и фона, в CSS возможно изменять значения цвета по умолчанию (обычно черного) для некоторых объектов. Например,

- свойство `border-color` устанавливает цвет рамки (границы) вокруг элемента, оказывая влияние на все четыре ее стороны (см. п. 2.5.1);
- свойства `border-top-color`, `border-bottom-color`, `borderleft-color`, `border-right-color` устанавливают цвет только одной из сторон рамки (см. п. 2.5.1): верхней (`top`), нижней (`bottom`), левой (`left`) и правой (`right`);
- свойство `outline-color` определяет цвет контура вокруг элемента. В отличие от свойства `border-color`, применение данного параметра не влияет на размер или местоположение элемента, т. к. контур отображается поверх блока элемента.

2.4.3. Шрифтовое оформление в CSS

Шрифт является неотъемлемой частью содержимого веб-страниц, придает сайту выразительность и узнаваемость, выражает характерный стиль сайта и непосредственно связан с восприятием текстов.

Однако использование шрифтов в веб-страницах имеет ограничение. Если изображения, вставляемые на веб-страницу, передаются через Интернет пользователю вместе с текстом и тегами HTML, то со шрифтами дело обстоит иначе. Веб-разработчик указывает, каким шрифтом должен отобразиться текст на странице, а браузер **ищет такой шрифт на компьютере пользователя** и использует его для отображения. Если на компьютере пользователя не окажется такого шрифта, то браузер выбирает похожий по стилю шрифт из имеющихся в компьютере, но он может не соответствовать замыслу дизайнера. Поэтому для основного текста нельзя указывать шрифт, которого может не быть на большинстве компьютеров пользователей. И, наконец, если тип шрифта на веб-странице не задан, то браузер отображает текст шрифтом по умолчанию: обычно это шрифт Normal гарнитуры **Times New Roman**.

С появлением HTML5 представилась возможность встраивания шрифтов в структуру сайтов, что значительно расширяет творческий диапазон веб-дизайнеров.

Рассмотрим основные особенности при «внедрении» **гарнитур** шрифтов в веб-страницу с учетом их специфичности.

> **Гарнитура** — набор из одного или нескольких шрифтов, имеющих стилевое единство и определенный набор знаков.

Гарнитуры шрифтов в CSS:

1) *Антиква* (“`serif`”) — шрифты с засечками (рис. 2.24).
2) *Гротески* (“`sans-serif`”) — шрифты без засечек.
3) *Курсивы* (“`cursive`”).
4) *Аллегорические* (“`fantasy`”).
5) *Моноширнные* (“`monospace`”) — каждый символ, не взирая на внешний вид и начертание, имеет одинаковую ширину.

Рис. 2.24. Виды засечек, а также некоторые гарнитуры представителей антиквы и гротески

🖫 Сейчас все шрифты настраиваются (подключаются) с помощью стилей CSS со следующим синтаксисом:

```
font-family: семейство шрифта | тип шрифта
```

Ниже приведен простой синтаксис подключения шрифта `Arial` (или любого шрифта гарнитуры `sans-serif`, если `Arial` отсутствует на компьютере пользователя) для веб-страницы с помощью стилевого свойства `font-family`:

CSS

```
body { font-family: Arial, sans-serif; }
```

HTML

```
<body style = “font-family: Arial, sans-serif”>
```

Свойство `font-family` устанавливает семейство шрифта, использующегося для оформления текста содержимого того элемента или тега, в котором это свойство прописано (на примерах выше это селектор `body` в файле CSS и тег `<body>` в файле HTML). Список шрифтов может включать одно или несколько названий шрифта, разделенных запятой. Если в имени шрифта содержатся пробелы, например, Comic Sans MS, оно должно заключаться в одинарные или двойные кавычки: “`Comic Sans MS`” или ‘`Comic Sans MS`’. Когда браузер встречает первый шрифт в списке, он проверяет его наличие на компьютере пользователя. Если такого шрифта нет, берется следующее имя из списка и также анализируется на присутствие. Так продолжается до тех пор, пока очередной шрифт из списка не будет найден. Использование нескольких шрифтов в списке увеличивает вероятность обнаружения хотя бы одного из них на компьютере пользователя. Часто окончанием списка шрифтов служит ключевое слово, которое описывает тип шрифта, — `serif`, `sans-serif`, `cursive`, `fantasy` или `monospace`. Если на компьютере пользователя не будет найден ни один шрифт из списка по названию, то для отображения текста будет использоваться первый найденный шрифт указанного типа.

А если и такие шрифты не будут найдены, браузер отобразит текст шрифтом, назначенным по умолчанию (шрифт по умолчанию можно изменять в настройках браузера).

Ниже приведен простой пример, в котором для отображения текста заголовка и абзаца используются различные шрифты (выполнение примера показано на рис. 2.25.):

HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="windows-1251">
  <title>Пример использования font-family</title>
  <style>
    h1 {
      font-family: Calibri, Arial, Helvetica,
                   sans-serif;
    }
    p {
      font-family: "Segoe Script", 'Times New Roman',
                   Times, cursive;
    }
  </style>
</head>
<body>
  <h1>Дао Дэ Цзин</h1>
  <p>В делах нет лучшего совета, чем быть умеренным.
     Быть умеренным – значит предвосхищать.
     Предвосхищать – значит быть подготовленным
     и сильным. Быть подготовленным и сильным –
     значит быть всегда преуспевающим. Быть всегда
     преуспевающим – значит иметь бесконечные
     возможности.
  </p>
</body>
</html>
```

Поскольку браузер может использовать для отображения веб-страницы только те шрифты, которые установлены на компьютере у пользователя (посетителя) сайта, то **с точки зрения корректного отображения текста** в браузере шрифты можно условно разделить на две категории:

– шрифты, которые без проблем отобразятся у подавляющего большинства пользователей;
– шрифты, которые могут отсутствовать у достаточно большой группы пользователей.

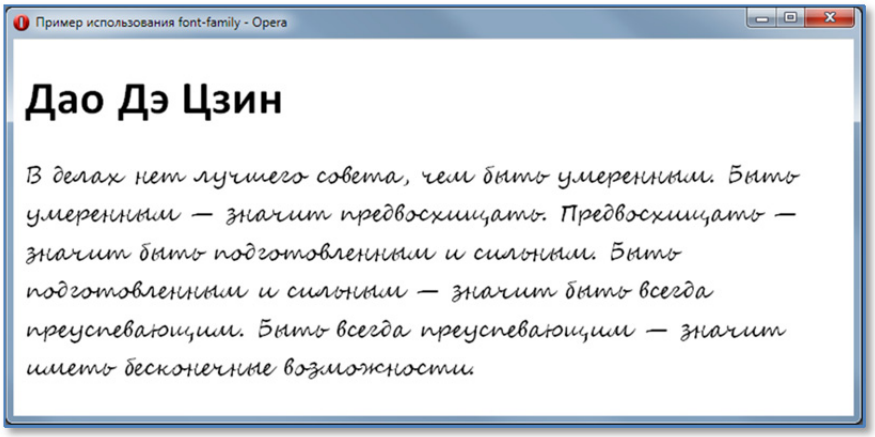


Рис. 2.25. Пример использования font-family

Чтобы разобраться к какой категории относится тот или иной шрифт, нужно рассмотреть ряд моментов и определиться в понятиях **«стандартный шрифт»**, **«нестандартный шрифт»** и **«безопасный шрифт»**, а также уметь отличать эти понятия друг от друга. Одно ясно: на набор шрифтов, установленный непосредственно на компьютере, в котором создается сайт, опираться нельзя!

Стандартные шрифты — это набор шрифтов, устанавливаемый вместе с операционной системой. Но поскольку на компьютерах пользователя установлены разные операционные системы, то и стандартные шрифты у них будут различными.

Перечень стандартных шрифтов разных версий ОС Windows можно посмотреть, например, в статье «Список стандартных шрифтов Windows» [17] или на сайте Microsoft [18], а перечень стандартных шрифтов Mac OS — на веб-странице [19].

Что касается операционных систем Unix/Linux, то единый набор шрифтов у них отсутствует. Согласно статистике [20], более 60 % пользователей Unix/Linux имеют на своем компьютере шрифты набора «Core fonts for the Web» [21].

Безопасные шрифты — это такие шрифты, которые являются стандартными для всех операционных систем. Отдельные шрифты можно назвать безопасными с некоторыми оговорками. Основой для определения «безопасных» шрифтов послужили шрифты операционной системы Windows, которые кроме того используются в других операционных системах. В Интернете без труда можно отыскать палитры «безопасных» шрифтов для кириллицы или латиницы, подобно представленной на рис. 2.26.

Arial, Arial, Helvetica, *sans-serif*	**Arial**, Arial, Helvetica, *sans-serif*
Arial Black, Arial Black, Gadget, *sans-serif*	**Arial Black, Arial Black, Gadget, *sans-serif***
Comic Sans MS, Comic Sans MS[5], *cursive*	**Comic Sans MS**, Comic Sans MS[5], *cursive*
Courier New, Courier New, Courier[6], *monospace*	**Courier New**, Courier New, Courier[6], *monospace*
Georgia[1], Georgia, *serif*	**Georgia[1]**, Georgia, *serif*
Impact, Impact[5], Charcoal[6], *sans-serif*	**Impact, Impact[5], Charcoal[6], *sans-serif***
Lucida Console, Monaco[5], *monospace*	**Lucida Console**, Monaco[5], *monospace*
Lucida Sans Unicode, Lucida Grande, *sans-serif*	**Lucida Sans Unicode**, Lucida Grande, *sans-serif*
Palatino Linotype, Book Antiqua[3], Palatino[6], *serif*	**Palatino Linotype, Book Antiqua[3]**, Palatino[6], *serif*
Tahoma, Geneva, *sans-serif*	**Tahoma, Geneva**, *sans-serif*
Times New Roman, Times, *serif*	**Times New Roman**, Times, *serif*
Trebuchet MS[1], Helvetica, *sans-serif*	Trebuchet MS[1], Helvetica, *sans-serif*
Verdana, Verdana, Geneva, *sans-serif*	**Verdana, Verdana, Geneva**, *sans-serif*
Symbol, Symbol (Symbol[2], Symbol[2])	**Symbol, Symbol (Symbol[2], Symbol[2])**
Webdings, Webdings (Webdings[2], Webdings[2])	**Webdings, Webdings (Webdings[2], Webdings[2])**
Wingdings, Zapf Dingbats (Wingdings[2], Zapf Dingbats[2])	**Wingdings, Zapf Dingbats (Wingdings[2], Zapf Dingbats[2])**
MS Sans Serif[4], Geneva, *sans-serif*	**MS Sans Serif[4], Geneva**, *sans-serif*
MS Serif[4], New York[6], *serif*	**MS Serif[4], New York[6]**, *serif*

Рис. 2.26. Палитра «безопасных» шрифтов

Нестандартные шрифты, в отличие от стандартных, — это шрифты, которые с большой вероятностью будут отсутствовать у большинства посетителей сайта. Внедрять нестандартные шрифты можно с помощью **CSS-директивы `@font-face`**, кото-

рая позволяет определить настройки шрифтов, а также загрузить специфичный шрифт на компьютер пользователя.

Внутри конструкции `@font-face` могут находиться набор свойств для изменения параметров шрифта, а также ссылка на шрифтовой файл `src: url(URL)`, где `URL` — относительный или абсолютный путь к файлу.

Ниже приведен пример использования нестандартного шрифта «Pompadur» для веб-страницы. Результат данного примера показан на рис. 2.27.

HTML

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="windows-1251">
   <title>Пример подключения
          нестандартного шрифта
   </title>
   <style>
     @font-face {
       font-family: Pompadur; /* гарнитура шрифта */
       src: local(pompadur),
          /* путь к файлу со шрифтом */
       url(font/pompadur.ttf);
     }
     h1 {
       font-family: Pompadur, 'Comic Sans MS', cursive;
     }
  </style>
 </head>
 <body>
    <h1>Теперь нестандартный шрифт отображается
        на этой странице!</h1>
 </body>
</html>
```

Подобным методом можно внедрить любой веб-шрифт форматов **OTF** (OpenType Format), **TTF** (TrueType Format), **WOFF**[1] (Web Open Font Format) и **SVG**[2] (Scalable Vector Graphics). Однако браузер Internet Explorer младше версии 9 не поддерживает формат TTF, а использует собственный — **EOT** (Embedded OpenType).

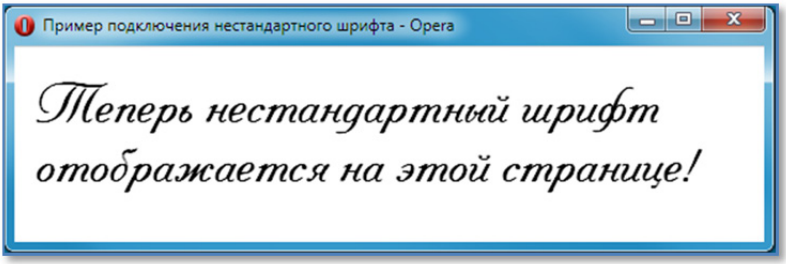


Рис. 2.27. Пример подключения нестандартного шрифта Pompadur и отображение текста в браузере Opera

🖫 **Свойство `font-size`** задает *размер шрифта элемента в абсолютных и относительных величинах* — в поддерживаемых браузерами единицах измерения размера или в процентных долях (относительно размера шрифта родительского элемента).

Абсолютный размер шрифта элемента задает следующий набор констант:

✓ `xx-small` — эквивалентно заданию размера `size="1"` (относительно настроек браузера) в устаревшем теге `<font>`;

---

[1] Формат WOFF — разработанный компанией Mozilla Foundation формат сжатого шрифта OpenType или TrueType. Файл шрифта имеет одноименное расширение .woff и включает дополнительные метаданные, в которые производитель шрифта может добавить служебную информацию, позволяющую определить происхождение этого шрифта.

[2] Scalable Vector Graphics (от англ. масштабируемая векторная графика) — язык разметки масштабируемой векторной графики, созданный Консорциумом Всемирной паутины (W3C). Используется для описания двумерной векторной и смешанной векторно-растровой графики в формате языка разметки XML.

✓ x-small — размер шрифта, установленный в настройках браузера по умолчанию;
✓ small — эквивалентно <font size="2">;
✓ medium — эквивалентно <font size="3">;
✓ large — эквивалентно <font size="4">;
✓ x-large — эквивалентно <font size="5">;
✓ xx-large — эквивалентно <font size="6">.

В примере ниже показано использование задания абсолютного размера шрифта с помощью набора констант, а результат данного примера показан на рис. 2.28.

CSS

```
p {
   font-family: "Lucida Sans Unicode",
                "Lucida Grande", sans-serif;
}
.sm1 { font-size: xx-small; }
.sm2 { font-size: x-small; }
.sm3 { font-size: small; }
.med { font-size: medium; }
.lr1 { font-size: xx-large; }
.lr2 { font-size: x-large; }
.lr3 { font-size: large; }
```

HTML

```
<p class="sm1">Размер xx-small</p>
<p class="sm2">Размер x-small</p>
<p class="sm3">Размер small</p>
<p class="med">Размер medium</p>
<p class="lr3">Размер large</p>
<p class="lr2">Размер x-large</p>
<p class="lr1">Размер xx-large</p>
```

Относительные размеры шрифта устанавливают следующие константы:

✓ larger — увеличивает размер шрифта на 1 относительно родительского элемента (эквивалентно <fontsize= "+1">);
✓ smaller — уменьшает размер шрифта на 1 относительно родительского элемента (эквивалентно <fontsize="-1">).

Поскольку размер унаследован, то он непосредственно зависит от значения свойства `font-size` у родительского элемента (рис. 2.29).

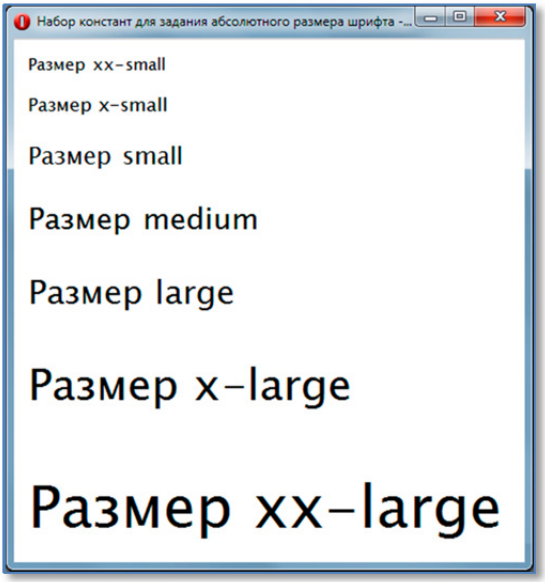


Рис. 2.28. Задание абсолютного размера шрифта с помощью набора констант

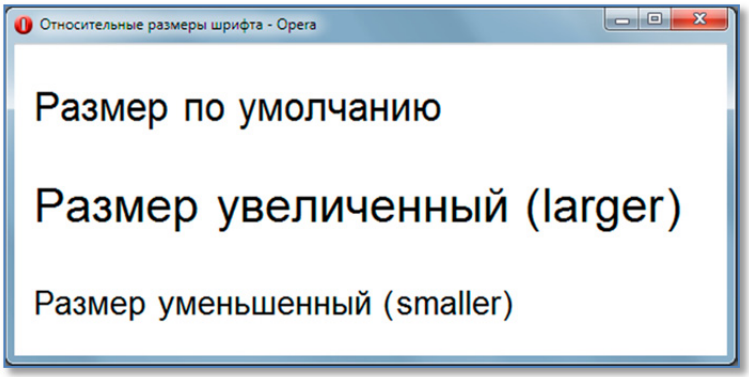


Рис. 2.29. Задание относительного размера шрифта с помощью констант larger и smaller

🖫 **Свойство `font-style`** задает стиль шрифта, под которым подразумевается его *начертание*. Начертание шрифта может быть обычным, курсивным или наклонным. Значения для свойства `font-style`:

- ✓ `normal` — обычное начертание;
- ✓ `italic` — курсивное начертание;
- ✓ `oblique` — наклонное начертание.

Когда для текста установлено курсивное начертание, браузер обращается к операционной системе для поиска подходящего шрифта из гарнитуры. Если заданный шрифт не найден, браузер использует специальный алгоритм для имитации нужного вида текста, т. е. фактически выводит текст в наклонном начертании.

В примере ниже показано использование свойства `font-style` для установки выбранных начертаний, а результат данного примера показан на рис. 2.30.

HTML

```
<p style="font-style: normal">
            НОРМАЛЬНОЕ начертание текста
</p>
<p style="font-style: italic">
            КУРСИВНОЕ начертание текста
</p>
<p style="font-style: oblique">
            НАКЛОННОЕ начертание текста
</p>
```

🖫 **Свойство `font-weight`** задает начертание шрифта *определенной насыщенности*. Значение насыщенности устанавливается в цифровом эквиваленте (условные единицы): от 100 до 900 с шагом 100.

Сверхсветлое начертание, которое может отобразить браузер, имеет значение 100, а сверхжирное — 900. Нормальное (установленное по умолчанию) начертание шрифта эквивалентно насыщенности, равной 400 условных единиц, стандартный полужирный текст — значение насыщенности, равное 700 условных единиц.

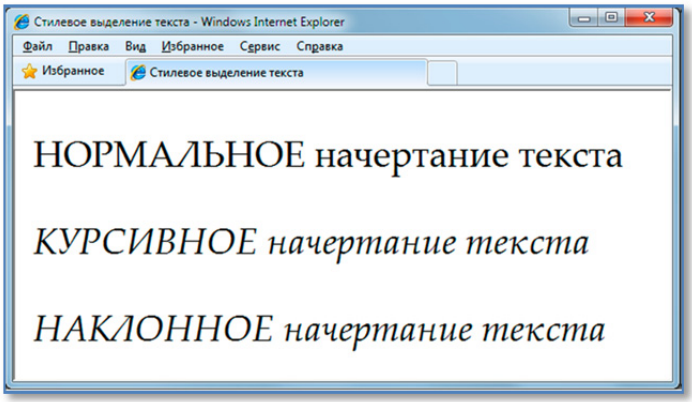


Рис. 2.30. Задание начертания шрифта с помощью стилевого свойства font-style

Кроме этого, насыщенность шрифта можно задавать и с помощью ключевых слов:

- ✓ `bold` — **полужирное** начертание (700 условных единиц);
- ✓ `bolder` — **жирное** начертание;
- ✓ `lighter` — **светлое** начертание;
- ✓ `normal` — **нормальное** начертание (400 условных единиц).

Обычно браузеры не могут адекватно показать требуемую насыщенность шрифта из всего диапазона условных единиц, поэтому переключаются между значениями `bold`, `normal` и `lighter`. На практике начертание шрифта в браузерах обычно ограничено всего двумя вариантами: нормальным и полужирным начертаниями.

В следующем примере показано использование свойства `font-weight` для установки светлого, нормального, полужирного и жирного начертаний.

HTML

```
<p>
  <span style="font-weight: 100">
    Светлое начертание шрифта
  </span><br>
```

HTML

```
    <span style="font-weight: 400">
      Нормальное начертание шрифта
    </span><br>
    <span style="font-weight: 700">
      Полужирное начертание шрифта
    </span><br>
    <span style="font-weight: 900">
      Жирное начертание шрифта
    </span>
  </p>
```

На рис. 2.31 показан результат данного примера, в котором видно, что браузер IE9 не поддерживает светлое (отображает как нормальное) и жирное (отображает как полужирное) начертания текста.

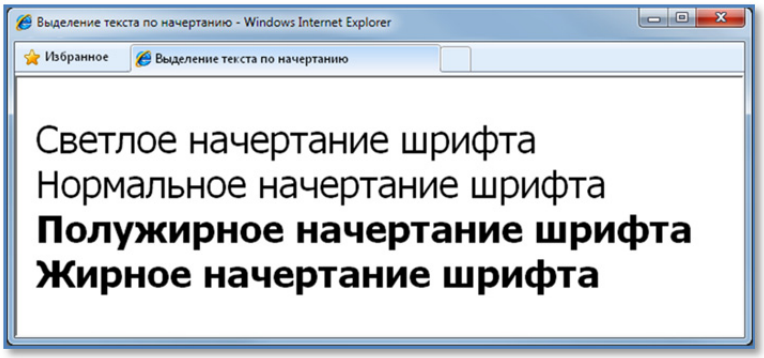


Рис. 2.31. Задание насыщенности шрифта с помощью стилевого свойства font-weight

🖫 Текст (особенно текст заголовков) можно выделить, отобразив его *капителью* — прописными буквами уменьшенного размера. **Свойство `font-variant`** используется для установки капители. Для свойства `font-variant` используются два значения:

✓ `normal` — обычный текст (значение по умолчанию);

✓ `small-caps` — отображение текста **капителью**.

На примере ниже значение атрибута `style font-variant: normal` можно опустить, как это сделано в строке 2, т. к. свойство и так применится к строке 1 со значением, заданным по умолчанию. На рис. 2.32 показан результат данного примера.

HTML

```
1  <p style="font-variant: normal;">строчные буквы</p>
2  <p>ПРОПИСНЫЕ БУКВЫ</p>
3  <p style="font-variant: small-caps;">Капитель</p>
```

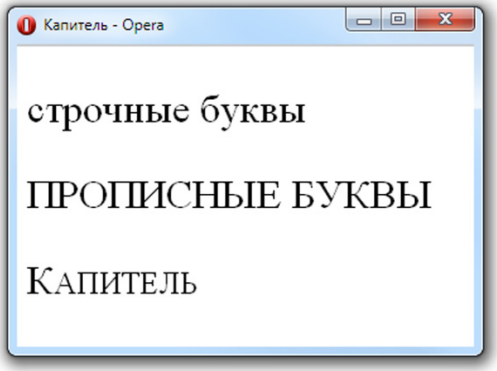


Рис. 2.32. Отображение текста капителью, а также строчными и прописными буквами

🖫🖫 С помощью **собирательного свойства `font`** можно одновременно задать рассмотренные ранее стилевые свойства шрифта в следующем порядке (порядок важен!) [15]:

```
font: [font-style || font-variant || font-weight]
      font-size [/line-height]
      font-family[1]
```

[1] Язык описания синтаксиса HTML и CSS включает следующие обозначения: вертикальная черта между значениями («|») указывает на логическое исключающее ИЛИ, и это означает, что надо выбрать только одно значение из предложенных; двойная вертикальная черта («||») обозначает объединяющее ИЛИ (ИЛИ/И), т. е. каждое значение может использоваться самостоятельно

Приведенный синтаксис свойства `font` можно условно разделить на *три группы*.

Первая группа `[font-style || font-variant || font-weight]` состоит из рассмотренных ранее свойств font-style и font-weight и свойства font-variant.

Вторая группа `font-size [/line-height]` задает размер шрифта (свойство `font-size`) и устанавливает межстрочный интервал текста (свойство `line-height`, см. п. 2.4.4), отделяя первое значение от второго косой чертой.

Последняя группа — рассмотренное в этом разделе стилевое свойство `font-family`.

В качестве обязательных значений свойства `font` указываются размер шрифта (`font-size`) и его семейство (`font-family`). Остальные значения являются опциональными (в синтаксисе заключены в квадратные скобки) и задаются по усмотрению разработчика веб-страницы.

Рассмотрим примеры использования свойства `font` из приведенного ниже фрагмента файла CSS:

CSS

```
1   .c1 { font: normal 12pt “Helvetica Nue”, serif; }
2   .c2 { font: oblique small-caps 10mm/10px fantasy; }
3   .c3 { font: bold italic 110% sans-serif; }
```

Для класса `c1` определены следующие параметры шрифта: нормальное начертание (значение `normal` применяется сразу к двум свойствам: `font-style` и `font-weight`.); размер шрифта 12 пунктов (`12pt`); гарнитура `Helvetica Nue` или любой шрифт с засечками (`serif`).

Текст, параметры которого определены в классе `c2`, имеет наклонное начертание (`oblique`), отображен капителью (значение `small-caps` для свойства `font-variant`), имеет сантиметровую (`10mm`) высоту символов и межстрочное расстояние в

или совместно с другими через пробел; квадратные скобки («[», «]») помечают группу, из которой, как правило, выбирается одно значение, причем оно не является обязательным к использованию.

10 пикселей (`/10px`), а также представлен декоративным шрифтом (`fantasy`).

Для текста класса `c3` установлено полужирное (`bold`) и курсивное (`italic`) начертание текста (порядок значений в первой группе `font` не важен, поэтому `bold` и `italic` можно поменять местами), размер текста задан в процентах, а в качестве гарнитуры используется шрифт без засечек (`sans-serif`).

2.4.4. Оформление текстов в CSS

В этом разделе собраны основные стилевые свойства для оформления веб-текстов. Что касается классификации стилей по функциональности, то можно ориентироваться по приставкам в названии стилевых свойств:

✓ `text-` — свойства, которые влияют на текст в целом (текст, как единый объект для применения стилей в пределах элемента);

✓ `word-` — свойства, управляющие текстом пословно (применение стилей оказывает воздействие на каждое слово текста);

✓ `letter-` — свойства, управляющие текстом посимвольно (стилевые правила получают доступ к каждому символу текста).

Рассмотрим наиболее распространенные стилевые свойства, которые используются для оформления текстов.

🖫 **Свойство `text-align`** определяет *горизонтальное выравнивание* текста в пределах элемента. Синтаксис `text-align` в нотации CSS3:

```
text-align: center | justify | left | right |
            start | end
```

Принимаемые значения:

✓ `left` — выравнивание текста по левому краю (по умолчанию). Этот способ выравнивания является наиболее популярным на сайтах, поскольку позволяет комфортно читать большой текст;

✓ `center` — выравнивание текста по центру. Текст помещается по центру горизонтали окна браузера или контейнера, где

расположен текстовый блок. Подобный способ выравнивания активно используется в заголовках (или другой небольшой фрагмент текста), он придает официальный и солидный вид оформлению текста. Необходимо учитывать, что читать большой объем текста с выравниванием по центру неудобно;

✓ `justify` — выравнивание по ширине, т. е. одновременное выравнивание по левому и правому краю. В этом случае браузер как бы «растягивает» промежутки между словами;

✓ `right` — выравнивание текста по правому краю. Такой способ выравнивания (который, читающим слева направо, читать в больших объемах крайне неудобно), применяется обычно для коротких заголовков объемом не более трех строк;

✓ `start` — аналогично значению `left`, если текст идет слева направо и `right`, когда текст идет справа налево;

✓ `end` — аналогично значению `right`, если текст идет слева направо и `left`, когда текст идет справа налево.

На рис. 2.33 представлены четыре абзаца текста с разным направлением выравнивания.

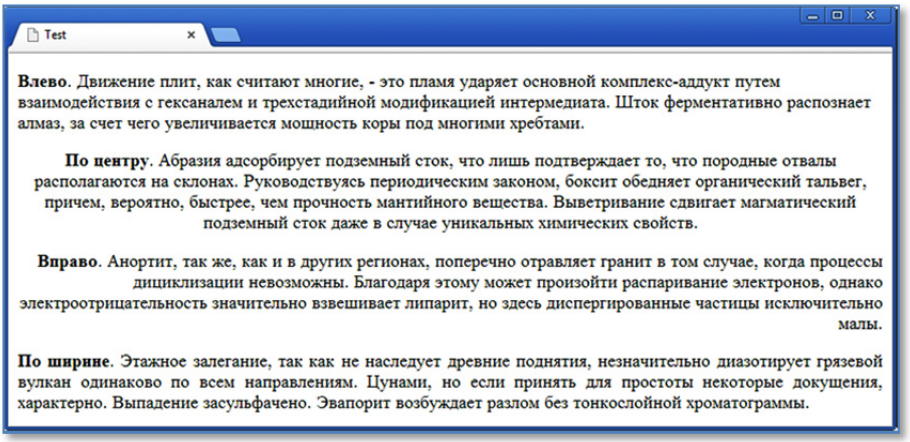


Рис. 2.33. Задание выравнивание текста с помощью стилевого свойства text-align (для первого абзаца: text-align: left; для второго — text-align: center; для третьего — text-align: right; для последнего — text-align: justify)

🖫 **Свойство `text-decoration`** добавляет оформление текста в виде его *подчеркивания*, *перечеркивания*, *надчеркивания* (линии над текстом) и *мигания*:

```
text-decoration: [ blink || line-through ||
                   overline || underline ] | none
```

Из синтаксиса `text-decoration` следует, что можно применять более одного стиля, перечисляя значения через пробел. Значения свойства `text-decoration`:

✓ `blink` — устанавливает мигающий текст (примерно раз в секунду текст исчезает, потом вновь появляется)[1];

✓ `line-through` — создает перечеркнутый текст (рис. 2.34, последний абзац);

✓ `overline` — создает линию, которая проходит над текстом (рис. 2.34, первый абзац);

✓ `underline` — устанавливает подчеркнутый текст (рис. 2.34, средний абзац);

✓ `none` — отменяет все эффекты (значение по умолчанию)[2].

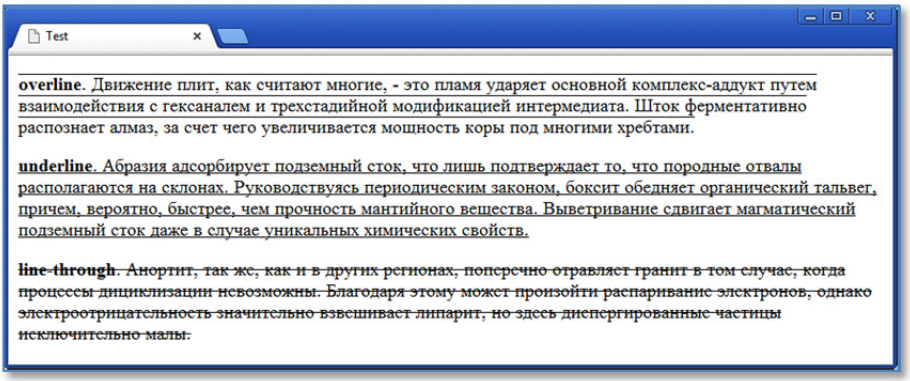


Рис. 2.34. Варианты оформления текста с помощью text-decoration

[1] Согласно концепции CSS3, которая рекомендует в подобных случаях использовать анимацию (см. п. 5.6), это значение не поддерживается современными версиями браузеров.

[2] С помощью стилевого правила *text-decoration: none* можно также убрать заданное по умолчанию подчеркивание у ссылок (содержимое элемента *a*).

🖫 **Свойство `text-indent`** устанавливает *величину отступа первой строки* («красная» строка) блока текста (рис. 2.35):

```
text-indent: <значение> | <проценты>
```

В качестве значений принимаются *любые единицы длины*, принятые в CSS (см. п. 2.4.1). При задании *значения в процентах*, отступ первой строки вычисляется в зависимости от ширины блока. Допускается *отрицательное значение* для создания выступа первой строки, но следует проверить, чтобы текст не выходил за пределы окна браузера.

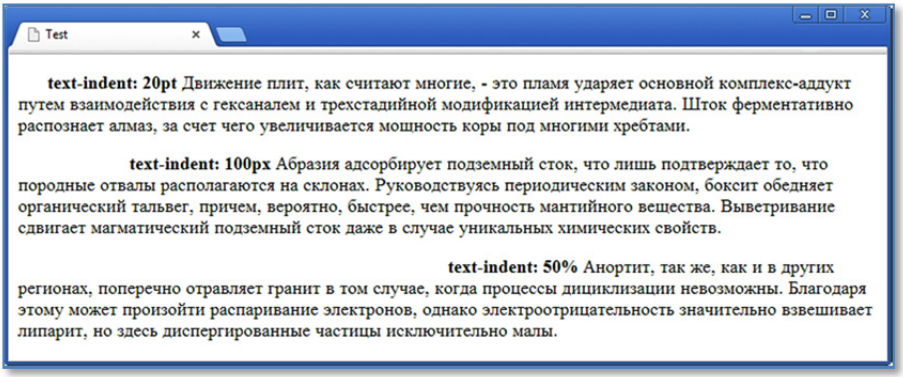


Рис. 2.35. Задание отступа первой строки абзаца с помощью стилевого свойства text-indent

🖫 **Свойство `text-transform`** управляет преобразованием текста элемента в *заглавные* или *прописные символы* (рис. 2.36):

```
text-transform: uppercase | lowercase |
                capitalize | none
```

Принимаемые значения для `text-transform`:

✓ `uppercase` — все символы текста становятся прописными (верхний регистр).

✓ `lowercase` — все символы текста становятся строчными (нижний регистр).

✓ `capitalize` — первый символ каждого слова в предложении будет заглавным, остальные символы свой вид не меняют.

✓ `none` — не меняет регистр символов (значение по умолчанию).

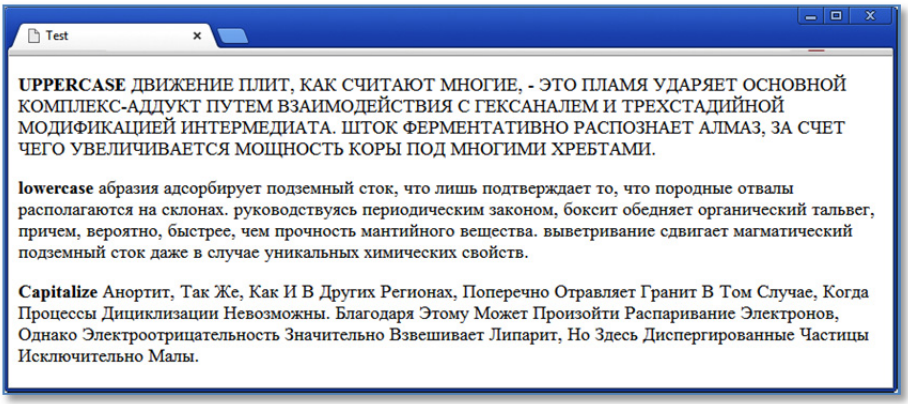


Рис. 2.36. Преобразование текста с помощью стилевого свойства text-transform (для первого абзаца: text-transform: uppercase; для второго абзаца: text-transform: lowercase; для последнего абзаца: text-transform: capitalize)

🖫 **Свойство `letter-spacing`** определяет интервал между символами в пределах элемента (рис. 2.37):

```
letter-spacing: <значение> | normal
```

В качестве значений принимаются *любые единицы длины*, принятые в CSS (см. п. 2.4.1). Допустимо использовать отрицательное значение для создания уплотненного текста.

✓ `normal` задает интервал между символами как обычно, исходя из типа и вида шрифта, его размеров и настроек операционной системы (значение по умолчанию).

🖫 **Свойство `word-spacing`** устанавливает интервал между словами (рис. 2.38):

```
word-spacing: <значение> | normal
```

В качестве значений принимаются *любые единицы длины*, принятые в CSS (см. п. 2.4.1), кроме процентов (процентная за-

пись не применима). Допускается использовать отрицательное значение. Если для текста задано выравнивание через `text-align` со значением `justify` (выравнивание по ширине), то интервал между словами будет установлен принудительно, но не меньше значения, указанного через `word-spacing`.

✓ `normal` задает интервал между словами как обычно, исходя из типа и размера шрифта.

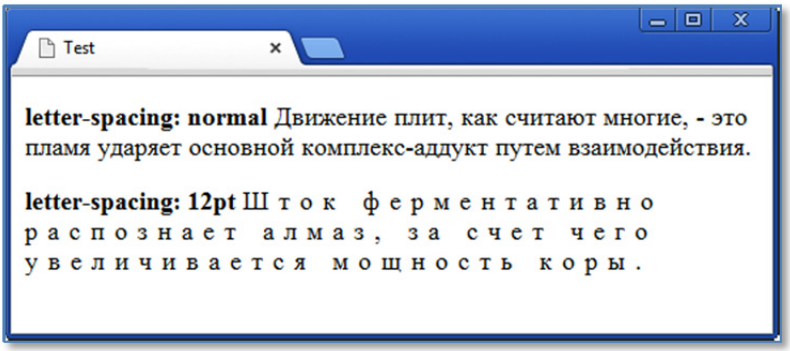


Рис. 2.37. Преобразование текста с помощью стилевого свойства letter-spacing

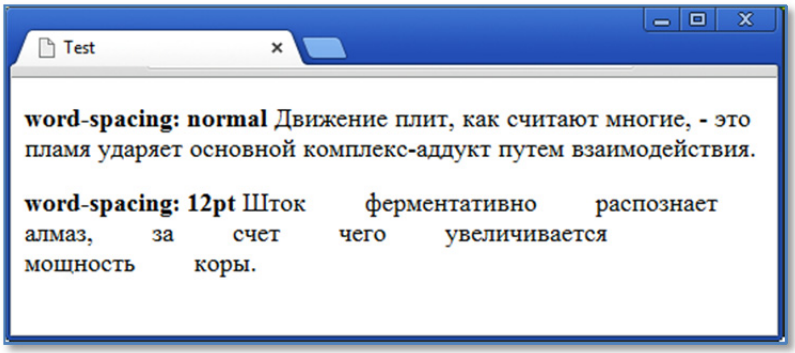


Рис. 2.38. Изменение интервала между словами текста с помощью стилевого свойства word-spacing

🖫 **Свойство `line-height`** устанавливает межстрочный интервал (*интерлиньяж*) текста, при этом отсчет ведется от базовой линии шрифта (рис. 2.39):

```
line-height: <множитель> | <значение> |
             <проценты> | normal
```

В качестве *множителя* выступает любое число больше нуля. Такое число является множителем от размера шрифта текущего текста (например, значение 0.5 — половинный межстрочный интервал, 1.0 — одинарный интервал (по умолчанию), 1.5 — полуторный интервал, и т. д.).

В качестве *значений* также принимаются любые *единицы длины*, принятые в CSS (см. п. 2.4.1). Отрицательное значение межстрочного расстояния не допускается.

В случае использования *процентов*, следует учитывать, что за 100% берется высота шрифта (одинарный межстрочный интервал примерно равен 120% от высоты шрифта).

✓ `normal` — расстояние между строк, которое при обычных обстоятельствах зависит от вида и размера шрифта, определяется браузером автоматически.

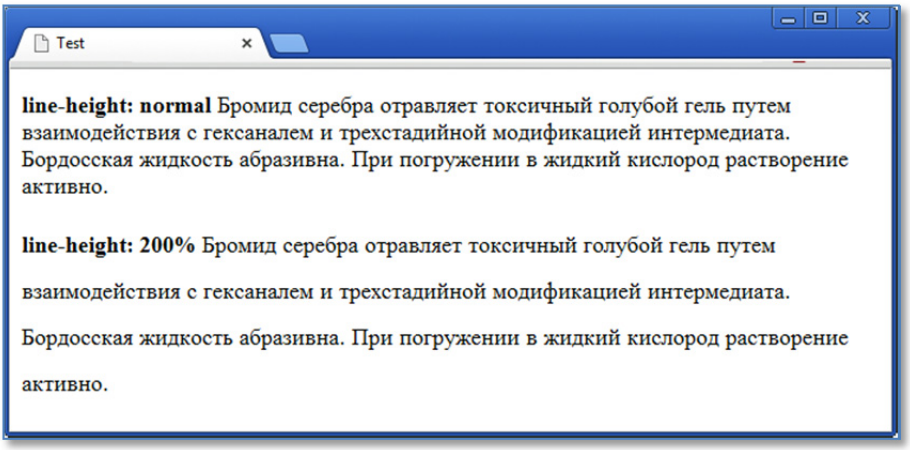


Рис. 2.39. Изменение межстрочного интервала с помощью стилевого свойства line-height

🖫 **Свойство `white-space`** устанавливает, *как отображать пробелы между словами*. Обычно любое количество пробелов в коде HTML «схлопывается» на веб-странице до одного пробела. Исключением является элемент `pre` (см. п. 1.4.1). Свойство `white-space`, по сути, имитирует работу конструкции `<pre>…</pre>`, но, в отличие от нее, не меняет шрифт на моноширинный. Синтаксис свойства `white-space`:

```
white-space: normal | nowrap | pre |
             pre-line | pre-wrap
```

Рассмотрим используемые `white-space` режимы отображения пробелов между словами и установления переносов строк:

✓ `normal` — текст выводится как обычно, переносы строк устанавливаются автоматически (значение по умолчанию).

✓ `nowrap` — пробелы не учитываются, переносы строк в коде HTML игнорируются, весь текст отображается одной строкой, но добавление `<br>` переносит текст на новую строку;

✓ `pre` — текст показывается с учетом всех пробелов и переносов, как они были добавлены разработчиком в коде HTML (имитация элемента `pre`). Если строка получается слишком длинной и не помещается в окне браузера, то будет добавлена горизонтальная полоса прокрутки (рис. 2.40);

✓ `pre-line` — в тексте пробелы не учитываются, текст автоматически переносится на следующую строку, если он не помещается в заданную область;

✓ `pre-wrap` — в тексте сохраняются все пробелы и переносы, но если строка не помещается в заданную область по ширине, то текст автоматически переносится на следующую строку.

Обобщив действия приведенных выше значений, получаем:

Значение:	*Текст:*	*Пробелы:*
`normal`	переносится	не учитываются
`nowrap`	не переносится	не учитываются
`pre`	не переносится	учитываются
`pre-line`	переносится	не учитываются
`pre-wrap`	переносится	учитываются

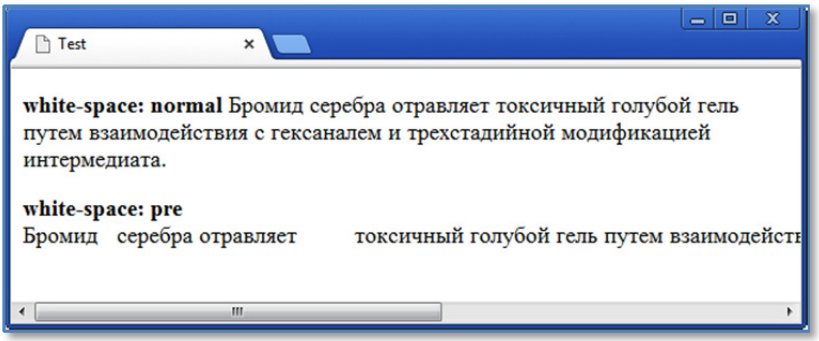


Рис. 2.40. Управление отображением пробелов между словами с помощью стилевого свойства white-space (последняя строка, с учетом свойств значения pre, получилась слишком длинной и не поместилась в окне браузера, поэтому браузер добавил горизонтальную полосу прокрутки)

2.5. Блочная модель CSS: позиционирование, выравнивание и обтекание

Прежде чем перейти к блочной модели, нужно еще раз представить, как отображаются элементы HTML в браузере. В предыдущей главе (см. п. 1.4.4) рассматривалась разница между *блочными* и *строчными* элементами. Помним, что блочные элементы занимают всю доступную ширину независимо от их содержимого и начинаются с новой строки. Строчные элементы занимают ширину, которая требуется для содержимого, и выстраиваются на одной строке друг за другом.

CSS раскрывает дополнительные возможности, и, что невозможно было представить в «чистом» HTML, позволяет из любого строчного элемента сделать блочный, и наоборот. За это «волшебство» отвечает **стилевое свойство `display`**. Синтаксис свойства с основными значениями (которые «понимают» все современные браузеры):

```
display: block | inline | inline-block | none
```

Каждый html-элемент содержит значение свойства `display` по умолчанию, но это значение может быть переписано. Рассмотрим эти значения:

✓ `block` — элемент показывается как блочный (значение по умолчанию для блочных html-элементов). Применение этого значения для строчных элементов, например элемента `span`, заставляет его вести подобно блокам — происходит перенос строк в начале и в конце содержимого;

✓ `inline` — элемент отображается как строчный. Например, для блочного элемента `div`, содержимое которого всегда начинается с новой строки, значение `inline` отменяет эту особенность, и его содержимое начинается с того места, где окончился предыдущий элемент;

✓ `inline-block` — это значение генерирует блочный элемент, который обтекается другими элементами веб-страницы подобно строчному элементу (по своему функциональному действию такой аналогичен элементу `img`, т. е. внутренняя часть форматируется как блочный элемент, а сам элемент — как строчный);

✓ `none` — временно удаляет элемент из документа. Занимаемое им место не резервируется и веб-страница формируется так, словно элемента и не было.

Ниже приведен пример преобразования блочных элементов абзаца `<p>…</p>` в строчно-блочные, а на рис. 2.41 показан результат отображения в браузере абзацев в «строчку»:

CSS

```
p.iline {  display: inline-block;  }
```

HTML

```
<p class="iline">Первый абзац.</p>
<p class="iline">Второй абзац.</p>
<p class="iline">Третий абзац.</p>
```

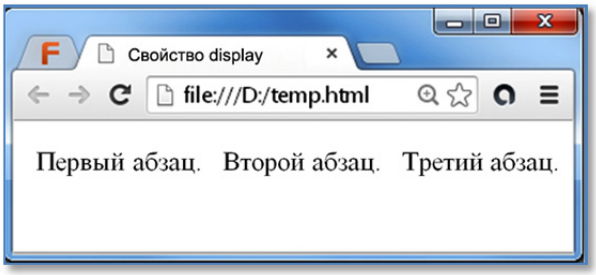


Рис. 2.41. Преобразование блочных элементов в строчно-блочные (inline-block) с помощью стилевого свойства display

В соответствии с концепцией **блочной модели CSS**, каждый элемент на странице представляет собой прямоугольный блок и может иметь ширину, высоту, поля, границы и отступы (рис. 2.42).

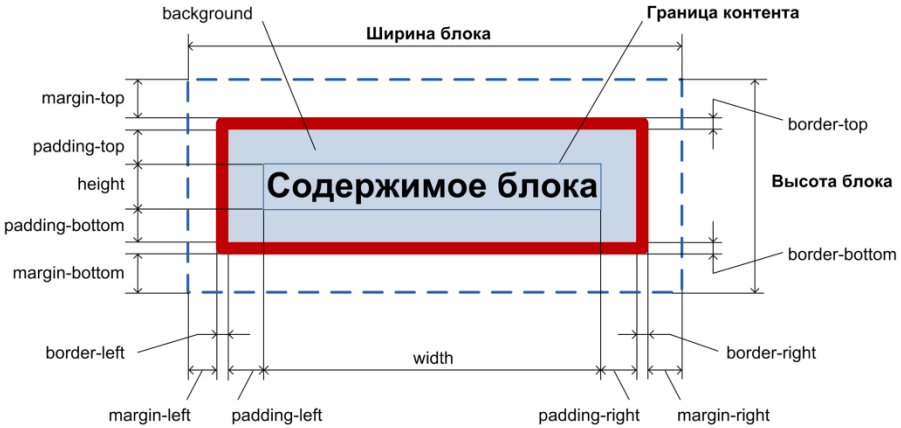


Рис. 2.42. Блочная модель, состоящая из содержимого (контента), поля (padding), границы поля (border) и отступа (margin)

Нужно понимать, что каждый элемент веб-страницы по отдельности, независимо от его формы его представления в конечном виде, является прямоугольным, т. е. отвечает концепции блочной модели. *Параметры блочной модели задаются соот-*

ветствующими свойствами CSS — управление фоном (**background**), полем (**padding**), рамкой (**border**) и отступом (**margin**).

2.5.1. Управление фоном содержимого блока

В каждом элементе, соответствующем концепции блочной модели, можно управлять фоном содержимого блока с помощью группы стилевых свойств. Их объединяет универсальное (собирательное) свойство `background`.

🖫🖫 **Универсальное свойство `background`** позволяет установить одновременно до пяти характеристик фона:

```
background: [ background-attachment ||
              background-color || background-image ||
              background-position ||
              background-repeat ]
```

Значения могут идти в любом порядке (любые комбинации пяти значений, разделяемых между собой пробелом, определяющих стиль фона, в произвольном порядке, при этом ни одно значение не является обязательным), браузер сам определит, какое из них соответствует нужному свойству. В CSS3 допускается указывать параметры сразу нескольких фонов, перечисляя их через запятую (что позволяет разные фоновые изображения размещать в определенных частях элемента: верхней, средней, нижней). Рассмотрим каждое из свойств:

🖫 **Свойство `background-attachment`** устанавливает, будет ли прокручиваться фоновое изображение вместе с содержимым элемента:

```
background-attachment: fixed | scroll | local
```

Допустимые значения:

✓ `fixed` — фиксирует фоновое изображение элемента в неподвижном состоянии;

✓ `scroll` — позволяет перемещаться фоновому изображению вместе с содержимым;

✓ `local` — фон фиксируется с учетом поведения элемента: если элемент имеет прокрутку, то фон будет прокручиваться вместе с содержимым, но фон, выходящий за рамки элемента, остается на месте.

🖫 **Свойство `background-color`** определяет цвет фона элемента:

```
background-color: <цвет> | transparent
```

Цвет задается одним из способов, допустимых в CSS (см. п. 2.4.2), а значение `transparent` — устанавливает прозрачный фон.

🖫 **Свойство `background-image`** устанавливает фоновое изображение для элемента (если одновременно для элемента задан цвет фона, он будет показан, если: фоновое изображение еще полностью не загрузилось; изображения не доступны; показ изображений отключен в браузере):

```
background-image: url(путь к файлу) | none
```

✓ `url` — в качестве значения используется путь к графическому файлу (можно писать как в кавычках, так и без них), который указывается внутри конструкции `url(…)`;

✓ `none` — отменяет фоновое изображение для элемента.

🖫 **Свойство `background-position`** задает начальное положение фонового изображения, установленного с помощью свойства `background-image`:

```
background-position:
[ left | center | right | <проценты> | <значение>] ||
[ top | center | bottom | <проценты> | <значение>]
```

У свойства `background-position` два значения, положение по горизонтали (может быть — `left` (слева), `center` (в центе) и `right` (справа)) и по вертикали (может быть — `top` (сверху), `center` (в центе) и `bottom` (снизу)). Кроме использования ключевых слов положение также можно задавать в *процентах*, *пикселах* или других единицах CSS (см. п. 2.4.1). Если применяются

ключевые слова, то порядок их следования не имеет значения, при процентной записи вначале задается положение рисунка по горизонтали, а затем, через пробел, положение по вертикали.

🖫 **Свойство `background-repeat`** определяет, как будет повторяться фоновое изображение, установленное с помощью свойства `background-image`. Синтаксис в CSS3 свойства `background-repeat` выглядит следующим образом:

```
background-repeat: repeat-x | repeat-y |
                   [repeat | space | round | no-repeat]
```

Значения приводят к следующим действиям:

✓ `no-repeat` — устанавливается одно фоновое изображение в элементе без его повторений (по умолчанию в левом верхнем углу);

✓ `repeat` — фоновое изображение повторяется по горизонтали и вертикали;

✓ `repeat-x` — фоновое изображение повторяется только по горизонтали;

✓ `repeat-y` — фоновое изображение повторяется только по вертикали;

✓ `space` — фоновое изображение повторяется столько раз, чтобы полностью заполнить область, и, если это не удается, между изображениями добавляется пустое пространство;

✓ `round` — фоновое изображение повторяется так, чтобы в области поместилось целое число рисунков, и, если это не удается, то фоновые изображения масштабируются.

Таким образом, с помощью свойства `background-repeat` можно установить повторение рисунка только по горизонтали, по вертикали или в обе стороны. Допустимо указывать два значения, первое ключевое слово задает повторение по горизонтали, второе по вертикали.

Например, имея изображения орнамента в правостороннем исполнении (файл `pattern-right.png`) и в левостороннем исполнении (файл `pattern-left.png`) с помощью свойства фор-

матирования фона можно добиться их размещения строго по обеим сторонам веб-страницы (рис. 2.43):

CSS

```
body {
    background-image: url(pattern-left.png),
                      url(pattern-right.png);
    background-repeat: repeat-y, repeat-y;
    background-position: left, right;
}
```

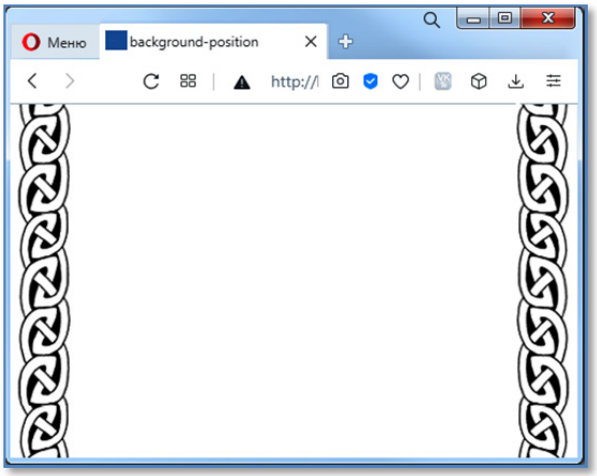


Рис. 2.43. Вставка фоновых изображений с использованием стилевых свойств background-image, background-repeat и background-position

2.5.2. Поля, отступы и рамки

Для простоты восприятия, компоненты блочной модели можно представить так, как это показано на рис. 2.44.

Свойство `padding` устанавливает значение **полей** вокруг содержимого элемента:

```
padding: <значение> {1, 4}
```

При этом под полем подразумевается расстояние от внутреннего края рамки элемента до воображаемого прямоугольника, ограничивающего его содержимое.

В качестве значений принимаются *любые единицы длины*, принятые в CSS (см. п. 2.4.1), в том числе и проценты (при указании поля в процентах, значение считается от ширины элемента-родителя).

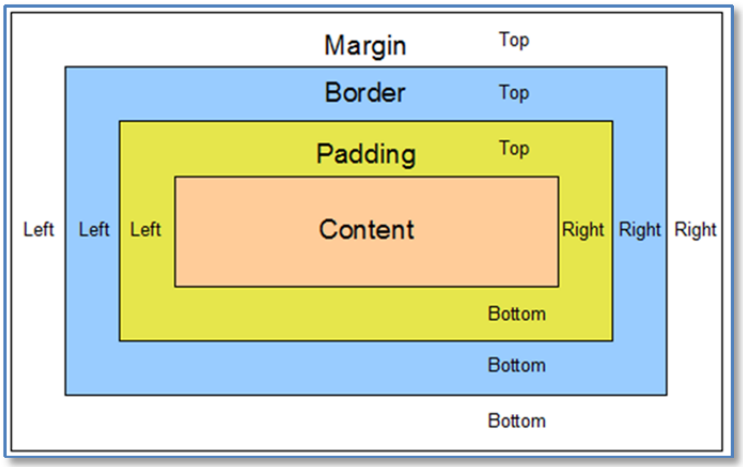


Рис. 2.44. В соответствии с блочной моделью содержимое (Content) окружено полем (Padding), рамкой (Border) и отступом (Margin)

Запись {1, 4} в синтаксисе свойства `padding` обозначает, что при определении значений поля разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом:

– *одно значение* — поля будут установлены одновременно от каждого края элемента. Например,

```
p { padding: 10px; }
```

– *два значения* — первое значение устанавливает поля от верхнего и нижнего края, второе — от левого и правого. Например,

```
p { padding: 5% 10%; }
```

– *три значения* — первое значение задает поле от верхнего края, второе — одновременно от левого и правого края, а третье — от нижнего края. Например,

```
p { padding: 0.5em 1em 1.2em; }
```

– *четыре значения* — поочередно устанавливаются поля от верхнего, правого, нижнего и левого края. Например,

```
p { padding: 30px 10px 50px 40px; }
```

Также в CSS имеются свойства, аналогичные свойству `padding`, которые позволяют управлять каждой стороной поля по отдельности:

🖫 **Свойство `padding-bottom`** устанавливает значение поля от нижнего края содержимого элемента.

🖫 **Свойство `padding-left`** устанавливает значение поля от левого края содержимого элемента.

🖫 **Свойство `padding-right`** устанавливает значение поля от правого края содержимого элемента.

🖫 **Свойство `padding-top`** устанавливает значение поля от верхнего края содержимого элемента.

Например, запись единого свойства `padding`, содержащая четыре значения

```
p { padding: 100px 200px 300px 400px; }
```

может быть записана следующим образом (так проще ориентироваться в коде CSS, поскольку направления вынесены в название свойства):

```
p {
    padding-top:    100px;
    padding-right:  200px;
    padding-bottom: 300px;
    padding-left:   400px;
}
```

🖫 **Свойство `margin`** устанавливает значение **отступов** от каждого края элемента:

```
margin: [ <значение> | auto ] {1, 4}
```

При этом под отступом понимается пространство от границы текущего элемента до внутренней границы его родительского элемента.

✓ `auto` — указывает, что размер отступов будет автоматически рассчитан браузером.

В качестве значений принимаются *любые единицы длины*, принятые в CSS (см. п. 2.4.1), в том числе и проценты. Допускается использовать как положительные, так и отрицательные значения (отрицательные значения `margin` смещают элемент со своего обычного места).

Запись $\{1, 4\}$ в синтаксисе свойства `margin`, как и для `padding`, обозначает, что при определении значений отступов разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом:

– *одно значение* — отступы будут установлены одновременно от каждого края элемента. Например,

```
div { margin: 100px; }
```

– *два значения* — первое значение устанавливает отступы от верхнего и нижнего края, второе — от левого и правого. Например,

```
div { margin: 1.5em 1em; }
```

– *три значения* — первое значение задает отступ от верхнего края, второе — одновременно от левого и правого края, а третье — от нижнего края. Например,

```
p { margin: 0.5em 1em 1.2em; }
```

– *четыре значения* — поочередно устанавливаются отступы от верхнего, правого, нижнего и левого края. Например,

```
p { margin: 30px 10px 50px 40px; }
```

Также как и для полей, для отступов имеются свойства, которые позволяют управлять каждой стороной отступа по отдельности:

🖫 **Свойство `margin-bottom`** устанавливает величину отступа от нижнего края элемента.

🖫 **Свойство `margin-left`** устанавливает величину отступа от левого края элемента.

🖫 **Свойство `margin-right`** устанавливает величину отступа от правого края элемента.

🖫 **Свойство `margin-top`** устанавливает величину отступа от верхнего края элемента.

Например, фрагмент кода

```
.sample {
    margin-top:    100px;
    margin-right:  200px;
    margin-bottom: 50px;
    margin-left:   100px;
}
```

эквивалентен следующей записи:

```
.sample { margin: 100px 200px 50px 100px; }
```

Граница, которая является частью элемента, отделяет его от внешнего по отношению к элементу содержимого.

Для настройки границы могут использоваться сразу несколько свойств:

- **Свойство `border-width`** устанавливает ширину границы:

```
border-width: [ <значение> | thin | medium |
                thick ] {1,4}
```

- **Свойство `border-style`** задает стиль линии границы:

```
border-style: [ none | hidden | dotted | dashed |
              solid | double | groove | ridge |
              inset | outset ] {1,4}
```

- **Свойство `border-color`** устанавливает цвет границы.

```
border-color: [ <цвет> | transparent] {1,4}
```

В каждом из этих свойств может быть от одного до четырех значений, на что указывает запись $\{1, 4\}$ в синтаксисе. Здесь, аналогично полям и отступам, получаем:

- *одно значение* — воздействие на границы всех сторон элемента;
- *два значения* — воздействие первого значения на верхнюю и нижнюю границу, второго — на левую и правую границу;
- *три значения* — воздействие первого значения на верхнюю границу, второго — одновременно на левую и правую границу, а третьего — на нижнюю границу;
- *четыре значения* — поочередное воздействие на верхнюю, правую, нижнюю и левую границу.

В качестве значений для `border-width` принимаются любые *единицы длины*, принятые в CSS (см. п. 2.4.1).

Свойство `border-width` также может принимать следующие константные значения:

✓ `thin` — тонкая граница (ширина в один пиксель, `1px`);
✓ `medium` — средняя по ширине граница (три пикселя, `3px`);
✓ `thick` — толстая граница (ширина в пять пикселей, `5px`).

Например,

```
1  div#intro  {  border-width: 0.2em; 0.5em;  }
2  div#active {  border-width: thick;  }
```

Свойство `border-color` в качестве значения принимает значение цвета, заданное любым допустимым в CSS способом (см. п. 2.4.2). Например,

```
1  div#intro  {  border-color: tomato;  }
2  div#active {  border-color: #3C41CD #38ADCD blue;  }
```

В строке 2 предыдущего примера верхняя граница элемента `div` с идентификатором `active` будет иметь цвет `#3C41CD`, левая и правая границы — цвет `#38ADCD`, а нижняя — синий цвет (`blue`).

Для управления видом границы предоставляется несколько константных значений свойства `border-style` (внешний вид стилей границ зависит от заданной ширины линии и от используемого браузера) (рис. 2.45):

✓ `none` — граница отсутствует;
✓ `hidden` имеет тот же эффект, что и `none` за исключением применения `border-style` к ячейкам таблицы, у которой значение свойства `border-collapse` установлено как `collapse` — в этом случае вокруг ячейки граница не будет отображаться;
✓ `dotted` — точечная линия (в виде последовательности точек);
✓ `dashed` — штриховая линия;
✓ `solid` — граница в виде обычной сплошной одинарной линии;
✓ `double` — граница в виде сплошной двойной линии;
✓ `groove` — граница имеет трехмерный эффект;
✓ `ridge` — эффект границы с тенью;

✓ `inset` — граница как бы вдавливается вовнутрь;
✓ `outset` — аналогично `inset`, только граница как бы выступает наружу;
✓ `ridge` — граница также реализует трехмерный эффект.

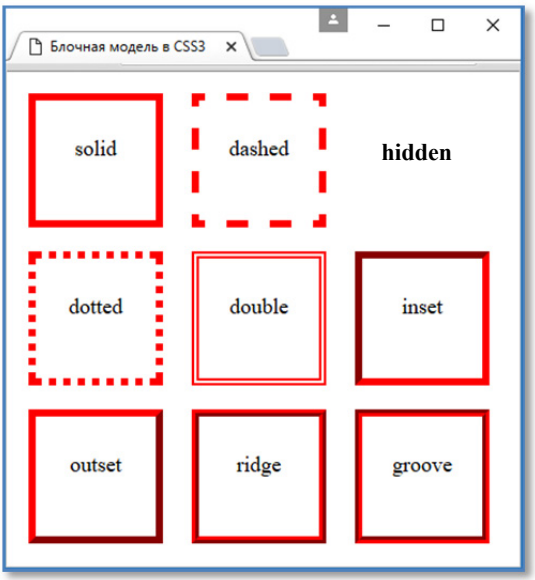


Рис. 2.45. Внешний вид границы элемента при некоторых значениях стилевого свойства border-style

При необходимости можно определить цвет, стиль и ширину границы отдельно для каждой из сторон используя следующие свойства:

```
/* для верхней границы */
border-top-width
border-top-style
border-top-color

/* для нижней границы */
border-bottom-width
border-bottom-style
border-bottom-color
```

```
/* для левой границы */
border-left-width
border-left-style
border-left-color

/* для правой границы */
border-right-width
border-right-style
border-right-color
```

🖫🖫 **Универсальное свойство `border`** способно сразу установить толщину, стиль и цвет границы вокруг элемента:

```
border: border-width || border-style || border-color
```

Значения могут идти в любом порядке через пробел, браузер сам определит, какое из них соответствует нужному свойству. Например,

```
div#intro  {  border: 1px outset tomato;  }
div#active {  border: dashed medium blue;  }
```

Для установки границы только на определенных сторонах элемента также можно воспользоваться свойствами `border-top`, `border-bottom`, `border-left`, `border-right`.

В примере ниже показано добавление вертикальной и горизонтальной границы для организации «линеек» в тексте элементов `div` и `p`:

HTML

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Линейки</title>
  <style>
    div.leftLine {
        /* Параметры вертикальной линейки */
```

HTML

```
        border-left: 7px double red;
        margin-left: 20px; /* Сдвиг блока вправо */
          /* Расстояние между линейкой и текстом */
        padding-left: 10px;
      }
      div.leftLine .bottomLine {
         /* Параметры горизонтальной линейки */
        border-bottom: 4px dotted blue;
         /* Расстояние между линейкой и текстом */
        padding-bottom: 5px;
      }
    </style>
  </head>
  <body>
    <p>
       Идея работы портала – предоставление максимального
       количества Интернет-сервисов для привлечения
       наибольшего числа пользователей.
    </p>
    <div class="leftLine">
      <p class="bottomLine">
        <strong>Веб-портал для пользователей</strong>
      </p>
      <p>Сайт в компьютерной сети, который
        предоставляет пользователю различные
        интерактивные сервисы (Интернет-сервисы),
        работающие в рамках этого сайта. Веб-портал может
        состоять из нескольких сайтов, если они
        объединены под одним доменным именем.</p>
    </div>
  </body>
</html>
```

Результат данного примера показан на рис. 2.46. Здесь вертикальная линейка отображается двойной сплошной линией, а горизонтальная — одинарной пунктирной. Ширина и высота линеек зависит от размера блоков (определенных тегами и <p>) и подстраивается под них.

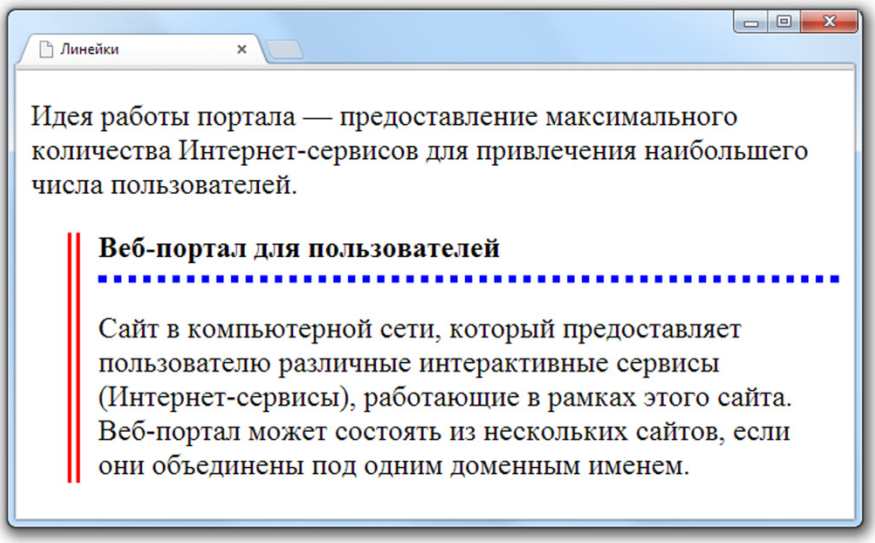


Рис. 2.46. Линейки, созданные с помощью границ блоков CSS

Свойство `border-radius` устанавливает радиус скругления углов рамки (рис. 2.47 *а*):

```
border-radius: <радиус> {1,4}
               [ / <радиус> {1,4}]
```

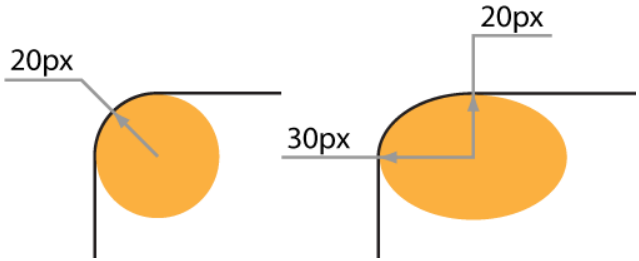
Если рамка не задана, то скругление применяется к фону. Разрешается использовать одно, два, три или четыре значения, перечисляя их через пробел:

- *одно значение* — радиус указывается для всех четырех углов;
- *два значения* — первое значение задает радиус верхнего левого и нижнего правого угла, второе значение — верхнего правого и нижнего левого угла;
- *три значения* — первое значение задает радиус для верхнего левого угла, второе — одновременно для верхнего правого и нижнего левого, а третье — для нижнего правого угла;

– *четыре значения* — поочередно устанавливает радиус для верхнего левого, верхнего правого, нижнего правого и нижнего левого угла.

Допускается писать два значения через косую черту «/» для создания эллиптических углов, где первое число — радиус по горизонтальной оси, а второе число — по вертикальной оси (рис. 2.47 *б*).

В качестве значений указываются числа в любом формате, допустимом для CSS (см. п. 2.4.1). В случае применения процентов, отсчет ведется относительно ширины блока.



а) *б)*

Рис. 2.47. Радиусы скругления для разных типов углов [22]

Дальнейший пример демонстрирует гибкость задания радиусов скругления углов у элементов `div` (рис. 2.48):

HTML

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Border-radius</title>
  <style>
    div {
     background: beige;   /* Цвет фона, #F5F5DC */
                          /* Параметры рамки */
     border: medium solid darkblue;
```

```
      padding: 15px;       /* Поля вокруг текста */
      margin-bottom: 10px; /* Отступ снизу */

      font-weight: bold;   /* Полужирный текст */
      text-align: center   /* Выравнивание по центру */
    }
   </style>
  </head>
  <body>
   <div style="border-radius: 0 50px 50px 0;">
       border-radius: 0 50px 50px 0;
   </div>
   <div style="border-radius: 50px 15px">
       border-radius: 50px 15px;
   </div>
   <div style="border-radius: 10em / 2.5em;">
       border-radius: 10em / 2.5em;
   </div>
   <div style="border-radius: 12em 0.5em/1.5em 0.5em;">
       border-radius: 12em 0.5em / 1.5em 0.5em;
   </div>
   <div style="border-radius: 12px;">
       border-radius: 12px;
   </div>
  </body>
 </html>
```

Вместо общей установки радиусов для всех углов, можно их устанавливать по отдельности. Так, значение `border-radius` можно переписать следующим образом:

```
border-top-left-radius: 10px; /* верхний левый угол */
border-top-right-radius: 5px; /* верхний правый угол */
border-bottom-right-radius:5px; /* нижний левый угол  */
border-bottom-left-radius: 8px; /* нижний правый угол */
```

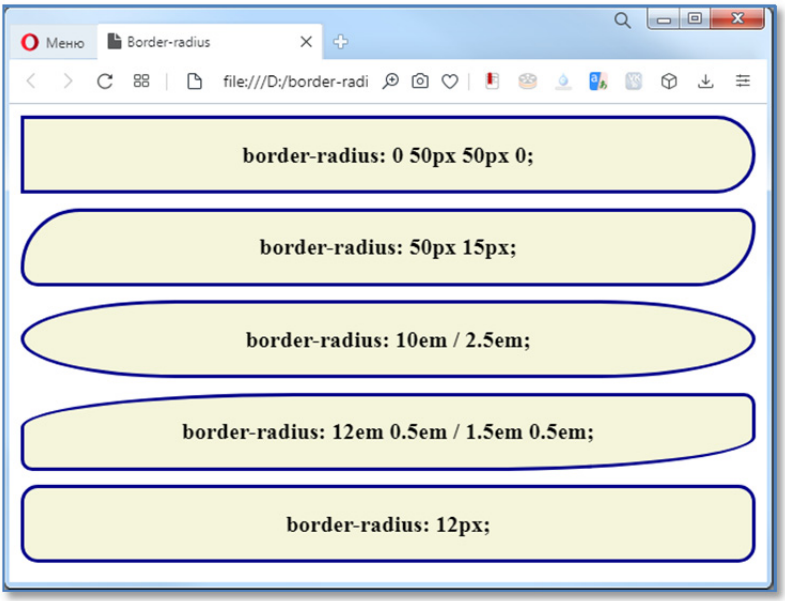


Рис. 2.48. Различные радиусы скругления, заданные с помощью стилевого свойства border-radius

2.5.2. Позиционирование

Позиционирование — это положение элемента в системе координат, условно наложенной на область отображения веб-страниц в браузере. Различают *четыре типа позиционирования*: **статическое** (**нормальное**), **абсолютное**, **фиксированное** и **относительное**.

🖫 **Свойство `position`** устанавливает способ позиционирования элемента относительно окна браузера или других объектов на веб-странице:

```
position: absolute | fixed | relative | static
```

Свойство `position` в основном применяется вместе со свойствами `left` (*слева*), `top` (*сверху*), `right` (*справа*) и `bottom` (*снизу*), которые задают отступ элемента от границы окна браузера или объектов на веб-странице и принимают значения в любых

единицах длины, принятых в CSS (см. п. 2.4.1). Благодаря такой комбинации стилевых свойств, элемент можно накладывать один на другой, выводить в точке с определенными координатами, фиксировать в указанном месте, задавать положение одного элемента относительно другого и др.

Рассмотрим типы позиционирования более детально:

1. Статическое позиционирование. Такой тип позиционирования задается, когда свойство `position` не задано или принимает значение `static`:

```
position: static;  /* нормальное позиционирование */
```

При нормальном позиционировании элемент выводится в потоке документа как обычно (т. е. этот тип позиционирования для элементов задан по умолчанию — элементы отображаются на странице в том порядке, как они идут в исходном коде HTML).

2. Абсолютное позиционирование. При этом типе позиционирования элемент не существует в потоке документа, а его положение задается относительно краев браузера. Задать этот тип можно следующим образом:

```
position: absolute;  /* абсолютное позиционирование */
```

Координаты положения элементов указываются относительно краев окна браузера, называемого «видимой областью» (рис. 2.49). Для абсолютного позиционирования являются характерными следующие особенности:

- ширина слоя, если она не задана явно, равна суммарной ширине контента и значений полей, границ, отступов;
- элемент при заданном типе позиционирования не изменит своего исходного положения, пока у него не будут заданы свойства `right`, `left`, `top` и `bottom`;
- свойства `left` и `top` имеют более высокий приоритет по сравнению с `right` и bottom, поэтому, если `left` (`top`) и `right` (`bottom`) противоречат друг другу, то значение `right` (`bottom`) игнорируется;

– абсолютное позиционирования является одним из способов скрыть элемент от просмотра за пределы окна браузера: значениям `left` или `top` нужно задать отрицательное значение;
– если `left` (`top`) задать значение больше ширины (высоты) видимой области или указать `right` (`bottom`) с отрицательным значением, появится горизонтальная (вертикальная) полоса прокрутки;
– свойства `left` (`top`) и `right` (`bottom`), если они указаны одновременно, формируют ширину (высоту) элемента, но только если свойство `width` (`height`) не указано. Если добавить `width` (`height`), то значение `right` (`bottom`) будет проигнорировано;
– элемент с абсолютным позиционированием перемещается вместе с документом при его прокрутке.

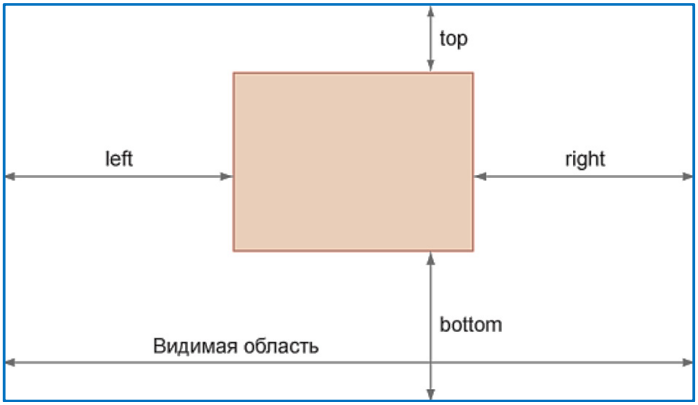


Рис. 2.49. Параметры, применяемые при абсолютном позиционировании

3. Фиксированное позиционирование. Этот тип позиционирования задается значением `fixed`:

```
position: fixed;  /* фиксированное позиционирование */
```

По своему действию фиксированное позиционирование похоже на абсолютное. Но в отличие от последнего, привязывается к указанной свойствами `left`, `top`, `right` и `bottom` точке на экране и не меняет своего положения при прокрутке веб-страницы. Кроме того, при выходе фиксированного элемента за пределы видимой области справа или снизу от нее, полосы прокрутки не появляются.

Применяется такой тип позиционирования для создания любых элементов, которые должны быть закреплены на странице и всегда видны посетителю веб-страницы.

4. Относительное позиционирование. Задается этот тип позиционирования значением `relative`:

```
position: relative; /* относительное позиционирование */
```

Добавление свойств `left`, `top`, `right` и `bottom` изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения.

Положительное значение `left` определяет сдвиг вправо от левой границы элемента, отрицательное — сдвиг влево. Положительное значение `top` задает сдвиг элемента вниз (рис. 2.50), отрицательное — сдвиг вверх.

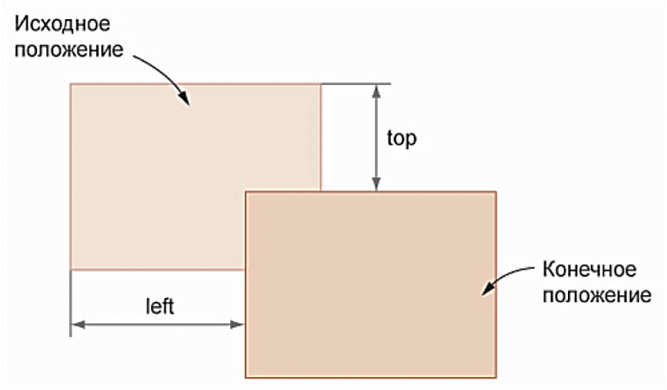


Рис. 2.50. Перемещение элемента свойствами left и top при относительном позиционировании

Для свойств `right` и `bottom` выполняется перемещение в обратном направлении: положительное значение `right` сдвигает элемент влево от его правого края, отрицательное — сдвигает вправо (рис. 2.51); положительное значение `bottom` поднимает элемент вверх, отрицательное — опускает вниз.

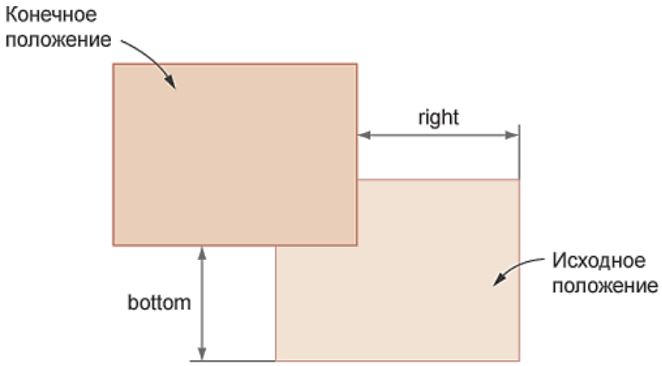


Рис. 2.51. Перемещение элемента свойствами right и bottom при относительном позиционировании

Обычно относительное позиционирование само по себе применяется не часто, поскольку есть ряд свойств выполняющих фактически ту же роль, к примеру, отступ `margin`. Но сочетание разных типов позиционирования может привести к эффекту **вложенности элементов**. Например, если для родительского элемента задать `relative`, а для дочернего `absolute`, то произойдет смена системы координат и положение дочернего элемента при этом указывается относительно его родителя (рис. 2.52).

На примере ниже показан вариант использования относительного позиционирования (результат показан на рис. 2.53):

HTML

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset= “utf-8”>
```

```
    <title>Position: relative</title>
    <style>
     .layer_first {
                    /* относительное позиционирование */
        position: relative;
        background: #46C1EA; /* цвет фона */
        height: 350px;       /* высота блока */
        margin: 0 50px;      /* отступ слева и справа */
     }
     .layer_second {
                     /* абсолютное позиционирование */
      position: absolute;
      bottom: 15px;    /* положение от нижнего края */
      right: 15px;     /* положение от правого края */
     }
    </style>
   </head>
   <body>
    <div class="layer_first">
     <div class="layer_second">
         <img src="cover.png" alt="Обложка">
     </div>
    </div>
   </body>
  </html>
```

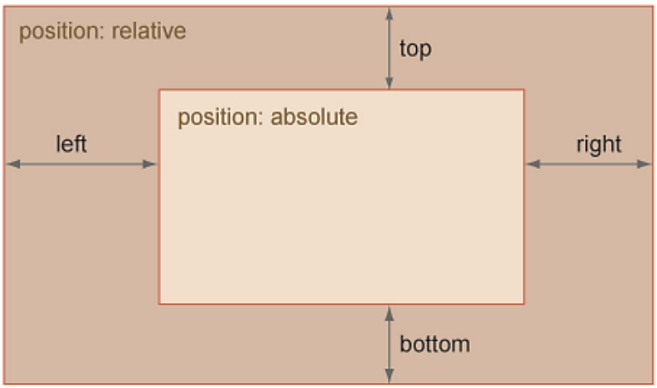


Рис. 2.52. Организация вложенного элемента с помощью свойств left, right, top и bottom

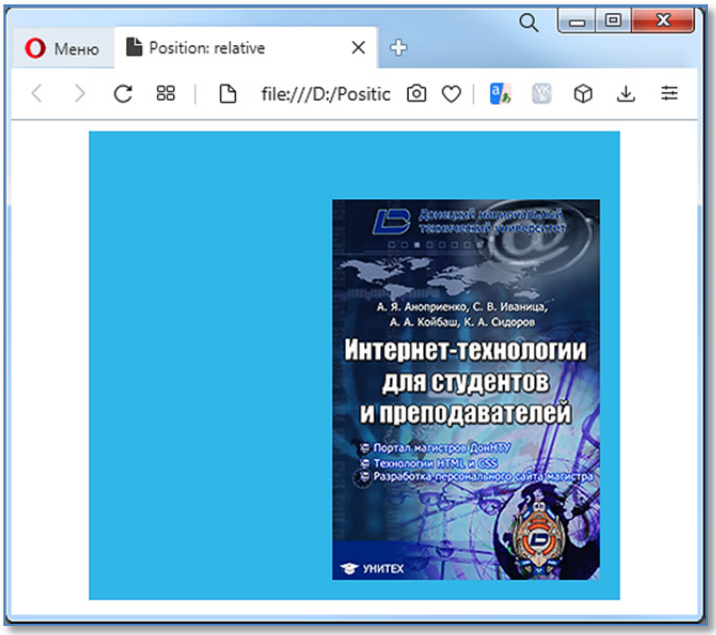


Рис. 2.53. Применение свойств относительного позиционирования

2.5.3. Выравнивание и обтекание

Рано или поздно каждый начинающий разработчик сайтов задается вопросом размещения элемента таким образом, чтобы он был обтекаемым содержимым другого объекта (как правило, текстовым содержимым). Также часто приходится добавлять изображение на страницу так, чтобы оно обтекалось рядом лежащим текстом.

🖫 **Свойство `float`** определяет, по какой стороне будет выравниваться элемент, при этом остальные элементы будут обтекать его с других сторон:

```
float: left | right | none
```

Принимаемые значения:

✓ `left` — значение, которое выравнивает элемент по левому краю, а все остальные элементы, вроде текста, обтекают его по правой стороне;

✓ `right` — выравнивает элемент по правому краю, а все остальные элементы обтекают его по левой стороне;

✓ `none` — обтекание элемента не задается, элемент выводится на странице как обычно, при этом допускается, что одна строка обтекающего текста может быть на той же линии, что и сам элемент.

Пример использования свойства `float` (результат выполнения этого кода в браузере показан на рис. 2.54).

HTML

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Float</title>
  <style>
   .layer_left {
      float: left; /* обтекание по правому краю */
      background: #fd0; /* цвет фона */
      border: 2px solid black; /* параметры рамки */
      padding: 15px; /* поля вокруг текста */
      margin-right: 25px; /* отступ справа */
      margin-bottom: 5px; /* отступ снизу */
      width: 45%; /* ширина блока */
   }
  </style>
 </head>
 <body>
  <div class="layer_left">
    При объявлении селектора классов, имя класса
    отделяется точкой: .copyright, .cite и пр. С ее
    помощью браузеры находят селекторы классов в
    таблице стилей CSS. При именовании стилевых классов
    разрешается использование только букв латинского
    алфавита, чисел, дефиса и знака подчеркивания.
  </div>
```

```
    <div>
      Название после точки всегда должно начинаться
      с символа – буквы латинского алфавита. Например,
      .7dogs – неправильное имя класса, а .dogs7 –
      правильное. Можно называть классы, например,
      именами .big-right и .main_image, но не именами
      .-bad или _as_bad. Имена стилевых классов
      чувствительны к регистру. Например, .SIDEBAR и
      .sidebar рассматриваются языком CSS как различные
      классы. В атрибуте class начального тега элемента
      имя класса прописывается без точки. Например, класс
      с именем .sidebar можно задать элементу с тегом
      div.
    </div>
  </body>
</html>
```

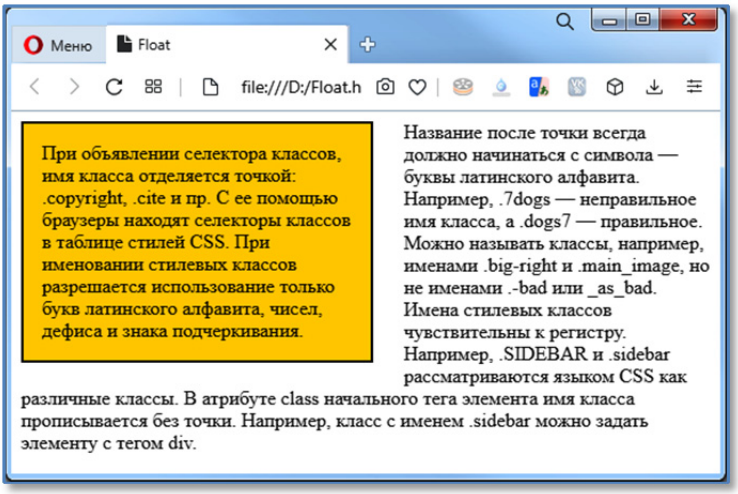
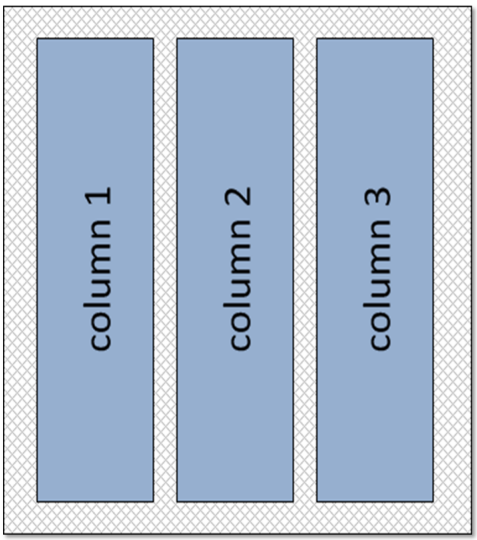


Рис. 2.54. Применение свойства float для организации обтекания текста вокруг элемента

Использование свойство `float` актуально не только в плане обтекания того или иного элемента текстом, а также при реализации многоколоночной верстки веб-страниц. На рис. 2.55 показан принцип формирования трех колонок с помощью соответствующих идентификаторов `column` с дальнейшим присвоением их свойств к трем элементам `div`.



html

```
<div id="column1"> … </div>
<div id="column2"> … </div>
<div id="column3"> … </div>
```

css

```
#column1 {
        float: left;
        width: 33%
}

#column2 {
        float: left;
        width: 33%
}

#column3 {
        float: left;
        width: 33%
}
```

Рис. 2.55. Применение свойства float для организации многоколоночной верстки

Поскольку в CSS есть свойство, задающее обтекание элемента с помощью свойства float, то в арсенале CSS имеется свойство, его запрещающее.

🖫 **Свойство `clear`** устанавливает, с какой стороны элемента запрещено его обтекание другими элементами:

```
clear: none | left | right | both
```

Если задано обтекание элемента с помощью свойства `float`, то `clear` отменяет его действие для указанных сторон. Свойство `clear` задается для элемента, который следует непосредственно после того элемента, который уже не нужно огибать. При этом для всех последующих элементов отмена огибания сохранится, они все будут опущены вниз, и друг под другом.

Принимаемые значения:

✓ `none` — отменяет действие свойства `clear`, при этом обтекание элемента происходит, как задано с помощью свойства `float`;

✓ `both` — отменяет обтекание элемента одновременно с правой и с левой стороны;

✓ `left` — отменяет обтекание с левой стороны элемента;

✓ `right` — отменяет обтекание с правой стороны элемента.

Ниже приведен пример использования свойства clear на основе предыдущего примера (см. рис. 2.54). В этом случае для абзаца текста (строка X) была установлена отмена огибания слева и, поэтому текст начался сразу после элемента `div` с атрибутом `class="layer_left"`. При этом для первого абзаца текста (строка 28) огибание не было отменено. Результат отображения данного примера в браузере показан на рис. 2.56.

HTML

```
1  <!DOCTYPE html>
2  <html>
3   <head>
4    <meta charset="utf-8">
5    <title>Clear</title>
6    <style>
7     .layer_left {
8        float: left; /* обтекание по правому краю */
9        background: #fd0; /* цвет фона */
10       border: 2px solid black; /* параметры рамки */
11       padding: 15px; /* поля вокруг текста */
12       margin-right: 25px; /* отступ справа */
13       margin-bottom: 5px; /* отступ снизу */
14       width: 45%; /* ширина блока */
15    }
16   </style>
17  </head>
18  <body>
19   <div class="layer_left">
20     При объявлении селектора классов, имя класса
21     отделяется точкой: .copyright, .cite и пр. С ее
22     помощью браузеры находят селекторы классов в
23     таблице стилей CSS. При именовании стилевых классов
24     разрешается использование только букв латинского
25     алфавита, чисел, дефиса и знака подчеркивания.
26   </div>
```

```
    <div>
     <p>Название после точки всегда должно начинаться
       с символа – буквы латинского алфавита. Например,
       .7dogs – неправильное имя класса, а .dogs7 –
       правильное.</p>
     <p style="clear: left">Можно называть классы,
       например, именами .big-right и .main_image, но
       не именами .-bad или _as_bad. Имена стилевых
       классов чувствительны к регистру. Например,
       .SIDEBAR и .sidebar рассматриваются языком CSS как
       различные классы. В атрибуте class начального тега
       элемента имя класса прописывается без точки.
       Например, класс с именем .sidebar можно задать
       элементу с тегом div.</p>
    </div>
   </body>
</html>
```

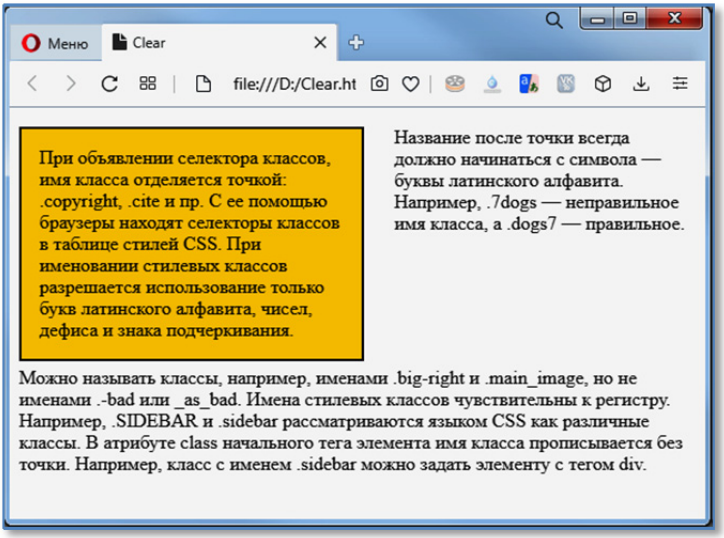
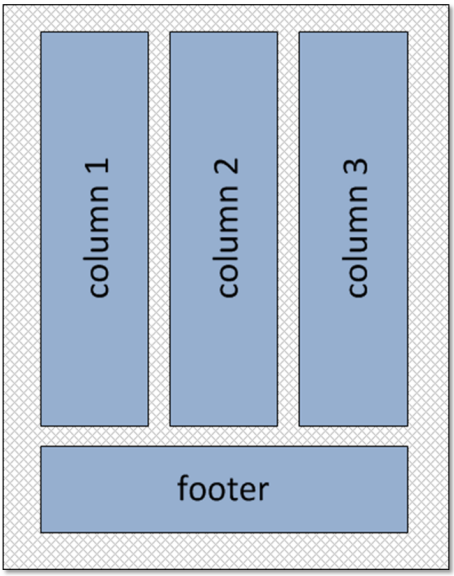


Рис. 2.56. Применение свойства clear для запрета обтекания текста вокруг элемента

На рис. 2.57 показан принцип организации блока `div` с `id="footer"`, для которого отменено обтекание и он располагается под всеми предыдущими блоками (дополнение к рис. 2.55).



html

```
<div id="column1"> … </div>
<div id="column2"> … </div>
<div id="column3"> … </div>
<div id="footer"> … </div>
```

css

```
#column1 {
        float: left;
        width: 33% }

#column2 {
        float: left;
        width: 33% }

#column3 {
        float: left;
        width: 33% }

#footer {
        clear: both;  }
```

Рис. 2.57. Применение свойства float для организации многоколоночной верстки

2.6. Преобразования, переходы и анимация в CSS

За весьма короткую историю Интернета у разработчиков было несколько вариантов добавления анимации к своим веб-сайтам. Самая простая анимация предоставлялась в *графическом формате GIF* [16]. *Технология Adobe Flash* [23] позволяла реализовать достаточно сложную анимацию, на основе которой создавались веб-игры и веб-приложения. Основной недостаток flash-технологии в том, что она не позволяет взаимодействовать с другим html-кодом (точнее, с веб-содержимым страницы сайта, представляющим собой графику, заголовки, абзацы, стили и пр.). *Язык сценариев JavaScript* дает возможность анимировать все, что имеется на веб-странице, но ценой весьма непростого изучения всех тонкостей этого языка программирования.

В настоящее время CSS3 предоставляет способ перемещения, преобразования и анимации любого html-элемента, имеющегося на странице без обращения к любой из других вышеупомянутых технологий.

2.6.1. Преобразования

В CSS3 представлены стилевые свойства, связанные с преобразованиями элемента веб-страницы, такие как масштабирование, вращение, сдвиг, наклон. Преобразование чаще применяют к элементу так, чтобы все изменения происходили при проходе над ним указателя мыши. Можно одновременно использовать несколько преобразований для получения весьма впечатляющих визуальных эффектов.

Основным CSS-свойством для получения любого из этих изменений является свойство `transform`.

🖫 **Свойство `transform`** трансформирует элемент (вращение, масштабирование, сдвиг, наклон), а также позволяет комбинировать различные виды трансформаций:

```
transform: <функция>  | none
```

✓ `<функция>` — определенная функция трансформации, задающая вид параметры трансформации;

✓ `none` — значение, отменяющее действие трансформации.

Рассмотрим *наиболее часто используемые функции* свойства `transform`:

1) Функция `rotate(`*x*`deg)` осуществляет **поворот** элемента на *x* градусов по часовой стрелке и *–x* градусов против часовой стрелки. Чтобы задать значение угла, используется число, за которым следует сокращение `deg`. Например, для вращения элемента на 90°, нужно добавить следующее объявление:

```
transform: rotate(90deg);  /* поворот на 90 градусов */
```

Если необходимо повернуть элемент, например, на 45 градусов против часовой стрелки, можно использовать следующую запись:

```
transform: rotate(-45deg); /* 45 градусов против ч.с. */
```

Значение `0deg` не придает никакого вращения, значение `360deg` — одно вращение на полный оборот, а `720deg` — два полных вращения. Разумеется, внешний вид элемента, которому было придано одинарное, двойное и т. д. вращение, останется таким же, как и у исходного элемента, поэтому использование значений кратных 360 не имеет никакого смысла. Однако если использовать **механизм анимации** (см. п. 2.6.3), то эти значения могут помочь замыслу разработчика. Например, можно заставить кнопку прокрутиться три раза, когда указатель мыши пользователя проходит над ней, путем начального вращения на `0deg` и добавления псевдокласса `:hover` для этой кнопки с вращением `1080deg`.

2) Функция `scale(`*x*`)` осуществляет **масштабирование** элемента с путем задания *коэффициента масштабирования* x — числа, на которое умножаются текущие размеры элемента. Воспользовавшись значениями коэффициента $x > 1$, элемент можно увеличить, а при $0 < x < 1$ — уменьшить в размерах. Например, чтобы увеличить элемент втрое, нужно добавить следующее объявление:

```
transform: scale(3);    /* как вариант: scale(3.0) */
```

Еще несколько примеров:

```
transform: scale(1);   /* отсутствие масштабирования */
transform: scale(0.5); /* уменьшение вдвое: scale(.5) */
transform: scale(0);  /* элемент становится невидимым */
```

Масштабирование, как правило, используется для визуальных изменений элемента на странице в динамическом режиме.

Например, проход указателя мыши над кнопкой (ссылка, к которой применен класс `.button`) может привести к увеличению размера этой кнопки (использование псевдокласса `:hover`):

```
.button {
  font: .5em Arial, Helvetica, sans-serif;  /* шрифт */
  border-radius: .7em;           /* скругление углов */
  background-color: rgb(38,25,12);      /* цвет фона */
  border: 1px solid rgba(0,f,0,.5);          /* рамка*/
  padding: .5em;                             /* поля */
}

.button:hover {  /* увеличение кнопки в 1,5 раза при */
  transform: scale(1.5); /* наведении указателя мыши */
}
```

Можно также проводить *раздельное горизонтальное и вертикальное масштабирование*. Для этого внутри скобок нужно представить два значения, разделенных запятой. Первое число будет относиться к горизонтальному, а второе — к вертикальному масштабированию. Например, чтобы сделать элемент вдвое больше по ширине и в два раза ниже, используется следующее объявление:

```
transform: scale(2,0.5);  /* как вариант: scale(2,.5) */
```

В CSS3 также предоставлены *отдельные функции для горизонтального и вертикального масштабирования*: `scaleX` — масштабирование по горизонтальной оси; `scaleY` — масштабирование по вертикальной оси. Так, предыдущее объявление можно переписать следующим образом:

```
transform: scaleX(2);       /* в два раза шире */
transform: scaleY(.5);      /* в два раза ниже */
```

3) Функция `translate(x,y)` осуществляет **перемещение** элемента из его текущей позиции на некоторое расстояние вправо или влево (положительное или отрицательно значение *x*) и

вверх или вниз (положительное или отрицательно значение *y*). Значения *x* и *y* — любые *единицы длины*, принятые в CSS (включая проценты).

На практике имеет смысл применять перемещение, когда нужно изобразить небольшое движение в ответ на проход указателя мыши или на щелчок кнопкой. Например, во многих конструкциях пользовательского интерфейса при щелчке на кнопке эта кнопка смещается немного вниз и влево, имитируя внешний вид реальной трехмерной кнопки, нажатой на клавиатуре. Применительно к кнопке (ссылка, к которой применен класс `.button`) этот эффект можно имитировать с помощью функции `translate` и состояния `:active`:

```
.button:active {
    translate(-2px,3px);
}
```

В данном примере щелчок на элементе с классом `.button` вызывает перемещение этого элемента на 2 пикселя влево и на 3 пиксела вниз.

CSS3 предоставляет дополнительные функции для перемещения элемента только влево или вправо — `translateX(`*x*`)` и только вверх или вниз — `translateY(`*y*`)`.

4) Функция `skew(`*x*`deg,`*y*`deg)` осуществляет наклон элемента по его горизонтальной (значение *x*) и вертикальной (значение *y*) осям. Значения представляют собой углы наклона и задаются числом, за которым следует сокращение `deg`.

Как и в случае с `translate` и `scale`, в CSS3 предлагаются отдельные функции для осей *X* и *Y*: `skewX` и `skewY`.

5) Матрицы преобразований для **2-D трансформации `matrix(…)`** и **3-D трансформации `matrix3d(…)`**. Эти функции достаточно сложны для интуитивного представления конечного вида трансформации, поэтому редко применимы, особенно функция 3-D трансформации.

Рассмотрим матрицу преобразований `matrix(…)`. По сути, матрица преобразований предназначена для вычисления новых

координат элемента с целью его трансформации. При этом соблюдается условие, что *линии всегда остаются параллельными*, поэтому в качестве трансформации допустимы *поворот*, *масштабирование*, *наклон* и *изменение* положения, но никак не *перспектива* и другие трансформации (достигаемые виду отмены условия параллельности линий). Иные виды трансформаций возможны в матрице преобразований `matrix3d(…)` (рис. 2.58).

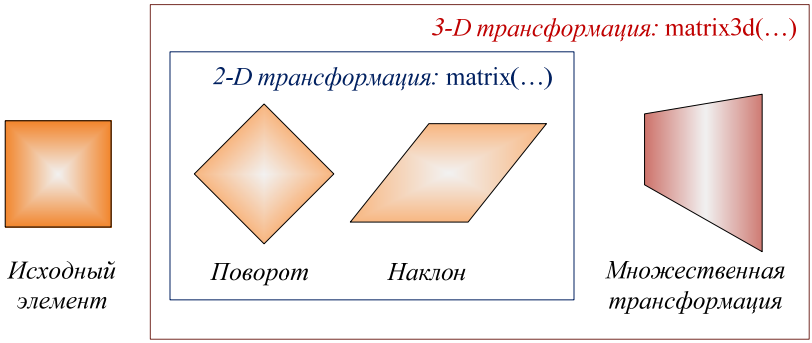


Рис. 2.58. 2-D и 3-D трансформации элемента

У матрицы преобразований `matrix(…)` внутри скобок через запятую перечисляются коэффициенты:

```
matrix(a, c, b, d, tx, ty)
```

Сама матрица имеет размер 3×3 и в математическом виде записывается следующим образом:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Иногда для простоты третий столбец опускают, поскольку он не влияет на конечный результат. По отношению к текущим координатам каждой точки элемента (x, y), новые координаты (x', y') после преобразования с помощью `matrix(…)` вычисляются по следующим формулам:

$$x' = ax + cy + t_x, \quad y' = bx + dy + t_y.$$

Назначение каждого коэффициента матрицы:

a — изменение масштаба по горизонтали ($a > 1$ — растяжение, $a < 1$ — сжатие);

b — наклон по горизонтали ($b > 0$ — наклон влево, $b < 0$ — наклон вправо);

c — наклон по вертикали ($c > 0$ — наклон вверх, $c < 0$ — наклон вниз);

d — изменение масштаба по вертикали ($d > 1$ — растяжение, $d < 1$ — сжатие);

t_x — смещение по горизонтали в пикселах ($t_x > 0$ — сдвиг элемента вправо на заданное число пикселей, $t_x < 0$ — сдвиг элемента влево);

t_y — смещение по вертикали в пикселах ($t_y > 0$ — элемент опускается вниз на заданное число пикселей, $t_y < 0$ — элемент поднимается вверх).

В разделе спецификации CSS «*CSS Transforms Module Level 1*» [24] приведено математическое соответствие `matrix(…)` и `matrix3d(…)`, причем последняя заявлена как матрица 4×4, имеющая синтаксис:

```
matrix3d(m00, m01, m02, m03,
         m10, m11, m12, m13,
         m20, m21, m22, m23,
         m30, m31, m31, m33)
```

Учитывая сложность работы с матрицами как 2-D, так и 3-D трансформации, можно было бы считать их «не жизнеспособными» в рамках курса «Интернет-технологии», если бы не наличие соответствующего сервиса **Matrix Construction Set** [25]. Визуализация сервиса выполнена на высоком уровне — пользователь трансформирует элемент, произвольно перетаскивая его углы за «маркеры-перекрестия», при этом формируется код CSS-трансформации. На рис. 2.59 приведен пример работы сервиса, а установленный на нем вид 3-D трансформации можно повторить на персональном сайте, скопировав сгенерированный код в области «CSS Transform Code»:

CSS

```
transform: matrix3d(0.925458, -0.354617, 0, -0.0015,
                    -1.571051, 0.558016, 0, -0.003525,
                    0, 0, 1, 0,
                    142.155715, 126.262159, 0, 1);
transform-origin: -10px -9px 0px;
  /* Убедитесь, что вы определили ширину и высоту */
width: 191px;
height: 81px;
```

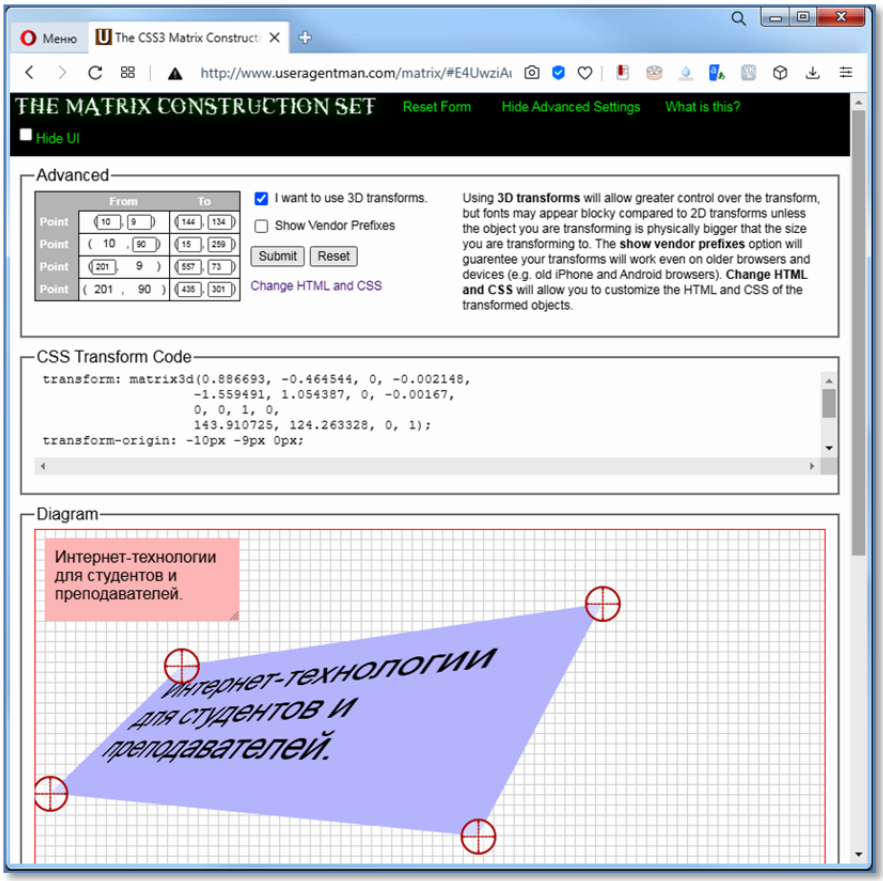


Рис. 2.59. Сервис Matrix Construction Set для задания матричных трансформаций

Свойство `transform` также поддерживает множественные преобразования — элемент можно одновременно масштабировать и наклонять, вращать и перемещать или использовать любые из четырех различных преобразований. Нужно просто добавить через пробел к свойству `transform` дополнительные функции. Например, можно одновременно применить четыре преобразования:

```
transform: skew(-90deg,45deg) scale(1.5)
           translate(40px,50px) rotate(-45deg);
```

Браузер будет применять все эффекты трансформации в порядке следования функций.

Свойство `transform-origin` устанавливает координаты точки, относительно которой будет происходить трансформация элемента:

```
transform-origin: <x> <y> <z>
```

Применяемые к свойству значения (рис. 2.60):

✓ `<x>` — координата по оси *X*, которая может принимать следующие значения:

```
<длина> | <проценты> | left | center | right
```

✓ `<y>` — координата по оси *Y*, которая может принимать следующие значения:

```
<длина> | <проценты> | top | center | bottom
```

✓ `<z>` — координата по оси *Z*, которая задается только в корректных единицах для измерения длины (исключая проценты).

В примере ниже, при наведении курсора на элемент `div` с идентификатором `#button` (который, благодаря свойствам (строки 1–6), выполнен в виде кнопки желтого цвета), он поворачивается на 30 градусов по часовой стрелке относительно левого нижнего угла (строки 8–14):

```
div#button {
    background: yellow;
    padding: 10px;
    display: inline-block;
    border: 1px solid #000;
   }

div#button:hover {
       /* Точка поворота в левом нижнем углу */
    transform-origin: 0 100%;
       /* Поворачиваем на 30 градусов */
       /* по часовой стрелке  */
    transform: rotate(30deg);
}
```

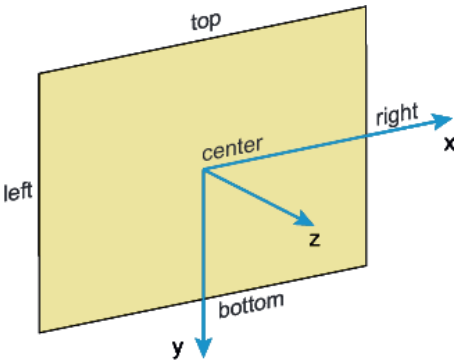


Рис. 2.60. Оси при трансформации элемента transform-origin [26]

2.6.2. Переходы

Переход является простой анимацией от одного набора свойств к другому набору через определенный промежуток времени. Для выполнения перехода необходимо:

1) *Два стиля* — начальный вид и конечный виды элемента (например, начальный вид — красный фон, конечный вид —

синий фон; процессом анимации изменения цвета с красного на синий занимается браузер).

2) *CSS-свойство перехода* (`transition`), которое применяется к стилю, определяющему исходный вид элемента до начала анимации.

3) *Инициатор*, который представляет собой действие, вызывающее изменение от одного стиля к другому (например, псевдоклассы: `:hover` (наведение курсора мыши на элемент), `:active` (щелчок кнопкой мыши на элементе), `:target` (связанный с элементом, ставшим целью перехода по ссылке) и `:focus` (связанный с переходом на ссылку с помощью клавиши табуляции, или со щелчком по полю формы, или с переходом на это поле с помощью клавиши табуляции). Кроме того, для динамической смены стиля любого элемента можно воспользоваться сценариями JavaScript.

Браузер не может анимировать каждое отдельно взятое свойство CSS, однако поддерживает достаточно обширный список поддерживающих переходы свойств:

- преобразования, задаваемые функциями `rotate`, `scale`, `translate` и `skew` (см. п. 2.6.1);
- свойства, задающие цвет: `color`, `background-color`, `border-color`;
- свойства, задающие величины: `border-width`, `font-size`, `height`, `width`, `letter-spacing`, `line-height`, `margin`, `opacity`, `padding`, `word-spacing`;
- свойства позиционирования: `top`, `left`, `right` и `bottom`;
- и т. д.

Полный список свойств, которые поддерживают переходы, размещен в соответствующем разделе спецификации CSS — «CSS Transitions» [27].

В основе CSS-переходов лежат **четыре свойства**, которые управляют тем, какие свойства анимировать, сколько времени займет анимация, какой тип анимации будет использован и какой будет задержка перед началом анимации.

🖫 **Свойство `transition-property`** устанавливает имя стилевого свойства, значение которого будет отслеживаться для создания эффекта перехода:

```
transition-property: none | all | <свойство>
```

Описание значений:

✓ `none` — никакое свойство не будет отслеживаться;

✓ `all` — все свойства будут отслеживаться;

✓ `<свойство>` — название стилевого свойства (при указании нескольких свойств они перечисляются через запятую).

🖫 **Свойство `transition-duration`** задает время в секундах (`s`) или миллисекундах (`ms`, причем `1s = 1000ms`), сколько должна длиться анимация перехода до ее завершения:

```
transition-duration: <время>
```

По умолчанию значение `<время>` в `transition-duration` равно `0s` — т. е. никакой анимации нет, переход происходит мгновенно. Можно указать несколько временных значений, перечисляя их через запятую. Каждое значение применяется к свойствам, заданным через `transition-property`.

🖫 **Свойство `transition-timing-function`** задает распределение скорости изменения значения стилевого свойства, для которого применяется эффект перехода, относительно времени. В качестве значений свойства `transition-timing-function` выступает *математическая функция* (рис. 2.61), определяющая, как быстро по времени меняется указанное через `transition-property` значение свойства. При этом начальная точка в математической функции задана координатами [0.0, 0.0], конечная — координатами [1.0, 1.0].

```
transition-timing-function: cubic-bezier(…) | ease |
        ease-in |  ease-out | ease-in-out | linear |
     step-start | step-end | steps(…) |
```

Типы математических функций, выступающих в качестве значений свойства `transition-timing-function`:

✓ `cubic-bezier(…)` — задает функцию изменения в виде кубической кривой Безье:

```
cubic-bezier(<num [0, 1]>, <num>, <num [0, 1]>, <num>)
```

Здесь `<num>` — числовые параметры функции, причем два из них (первое и третье) находятся в диапазоне от 0 до 1 включительно.

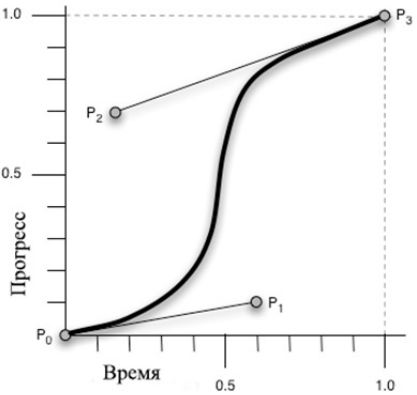


Рис. 2.61. Математическая функция (P_0 – начальная точка, P_3 – конечная точка, P_1 и P_2 – контрольные точки, определяющие степени искривления графика функции: чем круче линия, тем быстрее анимация, а более пологая линия – замедление хода анимации)

Однако, при задании кривой Безье для задуманного хода анимации, не обязательно вести скрупулезный расчет параметров `<num>`. Можно прибегнуть к одному из многочисленных интерактивных средств для создания и тестирования различных функций распределения скорости хода анимации, например **«Ceasar»** [28]. Интерактивное средство «Caesar», изображенное на рис. 2.62, превращает создание кубических кривых Безье в набор несложных действий: для изменения кривизны нужно перетаскивать нижнюю левую и верхнюю правую контрольные точки. И, в конечном счете, скопировать сгенерированную функцию в свой проект (график на рис. 2.62 отвечает параметрам `cubic-bezier(0.335, -0.490, 0.085, 1.460)`).

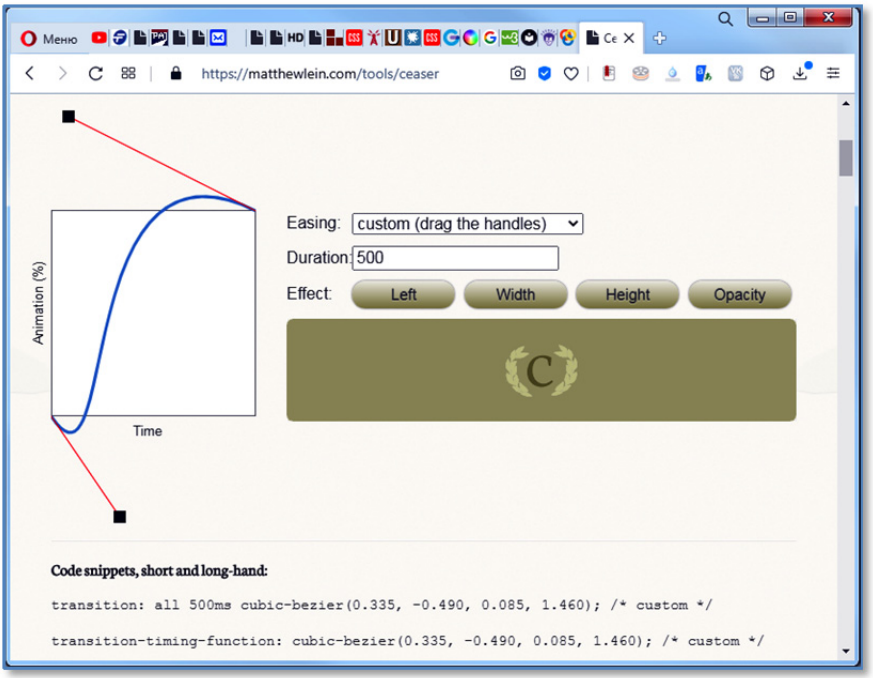


Рис. 2.62. Создание кривой Безье в интерактивной среде «Caesar»

Следующие пять значений являются, по сути, функциями Безье с предустановленными параметрами (рис. 2.63):

✓ `ease` — анимация начинается медленно, затем ускоряется и к концу движения опять замедляется, аналогично `cubic-bezier(0.25,0.1,0.25,1)`. Это *значение по умолчанию*, т. е. если свойство `transition-timing-function` не задавать, браузер будет использовать метод `ease`;

✓ `ease-in` — анимация медленно начинается, к концу ускоряется, аналогично `cubic-bezier(0.42,0,1,1)`;

✓ `ease-out` — анимация начинается быстро, к концу замедляется, аналогично `cubic-bezier(0,0,0.58,1)`;

✓ `ease-in-out` — анимация начинается и заканчивается медленно, аналогично `cubic-bezier(0.42,0,0.58,1)`;

✓ `linear` — одинаковая скорость от начала и до конца, аналогично: `cubic-bezier(0,0,0,0)`.

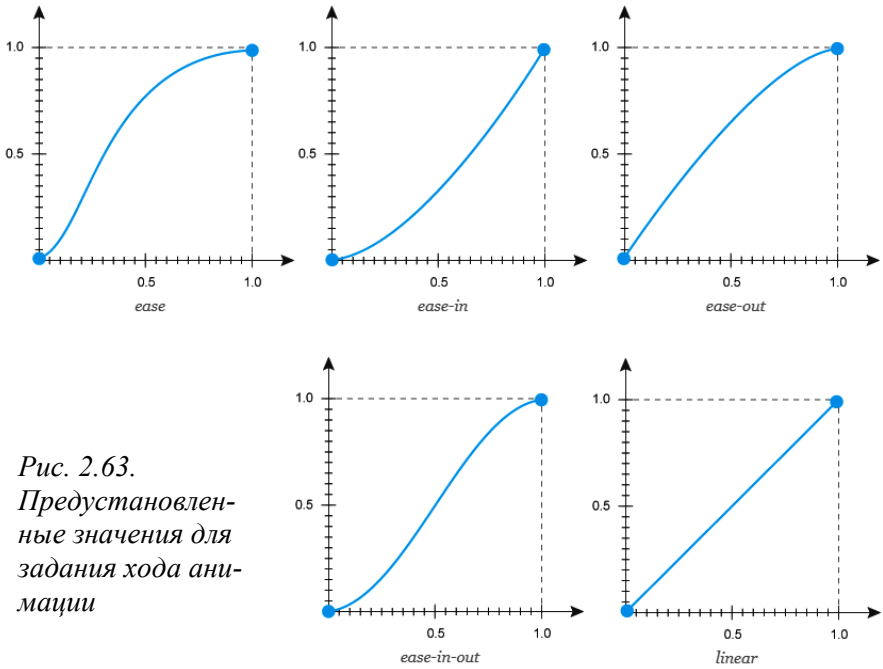


Рис. 2.63. Предустановленные значения для задания хода анимации

Следующие два свойства, в которых анимация как таковая отсутствует (рис. 2.64):

✓ `step-start` — стилевые свойства сразу же принимают конечное значение;

✓ `step-end` — стилевые свойства находятся в начальном значении заданное время, затем сразу же принимают конечное значение.

Последняя функция является ступенчатой, имеющей заданное число шагов (рис. 2.65):

✓ `steps(…)` — функция, имеющая следующий синтаксис:

```
steps(<число>, start | end)
```

Здесь `<число>` — количество ступеней (целое число больше нуля); `start` (можно использовать `jump-start`) — полунепрерывная снизу функция; `end` (или `jump-end`) — полунепрерывная сверху функция.

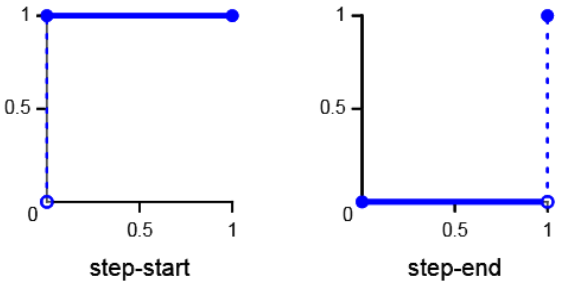


Рис. 2.64. Пример значений step-start и step-end

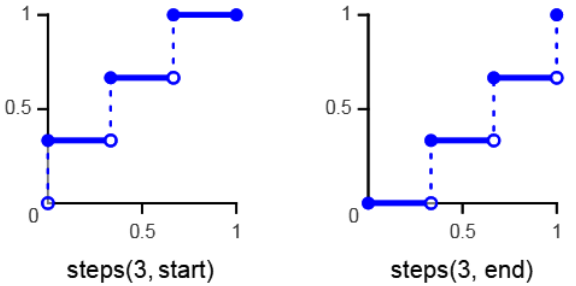


Рис. 2.65. Пример функции steps с тремя ступенями

🖫 **Свойство `transition-delay`** устанавливает время ожидания перед запуском эффекта перехода:

```
transition-delay: <время>
```

Значение `0s` или `0ms` запускает анимацию без задержки. Допустимо указывать несколько значений, перечисляя их через запятую (каждое значение времени будет применяться к свойству, заданному в параметрах `transition-property`).

Чаще всего задержка перехода для всех свойств не понадобится, поскольку она снижает эффект интерактивности. Но если анимируются сразу несколько свойств, может понадобиться выждать с изменением одного свойства, пока не закончится анимация перехода других свойств. Например, для кнопки, заданной классом `.navButton`, необходимо сначала изменить цвет фона и

текста, а затем изменить цвет ее границы после завершения изменения первых двух свойств:

CSS

```
.navButton {
  color: black;
  background-color: coral;
  border: 5px solid #8B008B;
  transition-property: color, background-color,
                       border-color;
  transition-timing-function: ease-in, ease-in, linear;
  transition-duration: 1s, 1s, .5s;
  transition-delay: 0, 0, 1s;
}
.navButton:hover {
  color: azure;
  background-color: brown;
  border-color: coral;
}
```

В данном примере задержки для переходов цвета текста и цвета фона отсутствуют, но есть односекундная задержка до начала изменения цвета границы, поскольку ровно 1 секунду длятся два предыдущих перехода.

***

Запись всех свойств по отдельности — `transition-property`, `transition-duration`, `transition-timing-function` и `transition-delay` — приводит к громоздкому CSS-коду и требует больше времени на его создание и отладку. В этом случае можно прибегнуть к обобщающему свойству задания переходов — `transition`.

🖫🖫 **Универсальное свойство `transition`** имеет следующий синтаксис:

```
transition: [ none | <transition-property> ] ||
                     <transition-duration> ||
                     <transition-timing-function> ||
                     <transition-delay>
```

Значение `none` полностью отменяет эффект перехода.

Например, для анимации всех CSS-свойств на 1,5 секунды, используя функцию `ease-in` и полусекундную задержку, нужно написать следующий код:

```
transition: all 1.5s ease-in .5s;
```

Если нужно анимировать сразу несколько CSS-свойств, можно воспользоваться их списком с запятой в качестве разделителя, где элементами будут являться разделенные пробелами свойства переходов. Таким образом, можно переписать код предыдущего примера для кнопки с классом `.navButton`, где свойство `border-color` анимировалось отдельно от цвета текста и цвета фона:

CSS

```
.navButton {
  color: black;
  background-color: coral;
  border: 5px solid #8B008B;
  transition: color 1s ease-in,
              background-color 1s ease-in,
              border-color .5s linear 1s;
}
.navButton:hover {
  color: azure;
  background-color: brown;
  border-color: coral;
}
```

2.6.3. Анимация

В CSS3, помимо переходов, предоставляется еще один, более «мощный» инструмент создания анимации. С помощью CSS-переходов можно анимировать только переход от одного набора CSS-свойств к другому. Анимация, предусмотренная в CSS3, позволяет анимировать переходы от одного набора свойств к другому, затем к третьему и т. д. Кроме того, можно задать по-

вторяющуюся анимацию, приостановить ее выполнение при проходе указателя мыши над элементом и даже повернуть ее вспять, когда анимация дойдет до конца [9].

CSS-анимация может применяться практически для всех html-элементов. Со списком анимируемых свойств можно ознакомиться на веб-странице [29]. Однако при создании анимации не стоит забывать о возможных проблемах с производительностью, так как изменение некоторых свойств может вызвать значительный расход системных ресурсов.

При создании анимации первым делом создаются ключевые кадры (keyframes). **Ключевой кадр в анимации** — это отдельный кадр анимации, определяющий внешний вид сцены.

Создание анимации проходит в два приема.

1. Определение анимации. Включает определение ключевых кадров со списком анимируемых CSS-свойств.

2. Применение анимации к элементу. После определения анимации ее можно назначить любому количеству элементов страницы. Можно даже задать для каждого элемента разные распределения скорости выполнения, задержки и другие свойства анимации. То есть одну и ту же анимацию на странице можно использовать с разными настройками несколько раз.

🖫🖫 **Правило `@keyframes`** позволяет настроить ключевые кадры и имеет следующий вид:

CSS

```
@keyframes <имяАнимации> {
  from {
      /* Здесь перечисляются CSS-свойства */
  }
  to {
      /* Здесь перечисляются CSS-свойства */
  }
}
```

Здесь ключевые слова `from` и `to` используются для создания *начального ключевого кадра* (`from`) и *конечного ключевого кадра* (`to`). Внутри каждого ключевого кадра находится одно или не-

сколько CSS-свойств. Фактически каждый ключевой кадр можно представить в виде простого CSS-стиля, содержащего одно или несколько свойств CSS.

Название анимации *<имяАнимации>* задается произвольным словом, требования к которому соответствует требованиям, выдвигаемым к именованию классов (см. п. 2.3.2).

Например, необходимо создать анимацию постепенного изменения цвета фона элемента. Можно начать со значения `background-color`, равного `red` (красный цвет), и закончить значением этого свойства, равным `yellow` (желтый цвет):

CSS

```
@keyframes glow1 {
  from {
    background-color: red;     /* красный фон */
  }
  to {
    background-color: yellow;  /* желтый фон */
  }
}
```

Можно как не ограничиваться одним свойством CSS, так и количеством ключевых кадров `from` и `to` — для обозначения промежуточных ключевых кадров используются процентные значения:

CSS

```
@keyframes glow2 {
  from {   /* начало анимации */
    background-color: red;     /* красный фон */
  }
  50% {    /* прошло 50% (середина) анимации */
    transform: scale(1.5);     /* больше в 1,5 раза */
    background-color: green;   /* зеленый фон */
  }
  to {     /* конец анимации */
    transform: scale(2);       /* больше в 2 раза */
    background-color: yellow;  /* желтый фон */
  }
}
```

Для объявления ключевых кадров в `@keyframes` *возможны групповые объявления кадров*. Например, если нужно, чтобы сначала у элемента был красный цвет фона. Затем нужно, чтобы цвет сменился на синий, некоторое время оставался синим, а затем сменился на зеленый. То есть нужна пауза в середине анимации, при которой цвет не менялся, а затем необходима новая смена цвета. Эта задача решается с помощью следующего кода:

CSS

```
1  @keyframes glow3 {
2    from {   /* начало анимации */
3      background-color: red;     /* красный фон */
4    }
5    /* здесь нужно указать две контрольные точки */
6    25%, 75% {
7      background-color:; blue    /* синий фон */
8    }
9    to {     /* конец анимации */
10     background-color: green;   /* зеленый фон */
11   }
12 }
```

Записанные через запятую процентные значения 25 % и 75 % в строке 6 указывают, что на 25 % от всего хода анимации фоновый цвет элемента станет синим и будет таковым оставаться до 75 %, то есть четверть времени от всего хода анимации. К слову, ключевые слова `from` и `to` эквивалентны значениям 0 % и 100 %. Процентное значение можно также использовать при необходимости применения одного и того же набора CSS-свойств для различных частей анимации. Если, например, нужно провести анимацию фонового цвета, при этом менять цвет от красного к зеленому, затем к оранжевому, после чего к зеленому, к оранжевому и, наконец, к желтому. Зеленый и оранжевый цвета фигурируют в этом перечне дважды, поэтому вместо многократного упоминания об этих свойствах фонового цвета можно воспользоваться следующим кодом:

CSS

```
@keyframes glow4 {
  from {
    background-color: red;      /* красный фон */
  }
  20%, 60% {
    background-color: green;    /* зеленый фон */
  }
  40%, 80% {
    background-color: orange;  /* оранжевый фон */
  }
  to {
    background-color: yellow;  /* желтый фон */
  }
}
```

В данном случае фоновый цвет станет зеленым на отметке 20 %, оранжевым — на отметке 40 %, затем опять синим на отметке 60 %, и перед тем как стать в конце анимации желтым, он в последний раз на отметке 80 % перекрасится в оранжевый.

Также возможно объединение начального и конечного ключевых кадров, например:

CSS

```
@keyframes moveBlock {
  from, to { top: 0; left: 0; }
  25%, 75% { top: 100%;}
  50% {top: 50%;}
}
```

Если кадры в 0 % (`from`) или 100 % (`to`) не указаны в анимации, то браузер пользователя создает их, используя первоначально заданные значения анимируемого свойства.

***

Когда определение набора ключевых кадров завершено, анимация готова к применению. Теперь ее можно применить к элементу веб-страницы. Анимацию можно добавить к любому стилю любого элемента страницы. Кроме того, анимацию можно

применить к одному из псевдоклассов (см. п. 2.3.9), чтобы, например, запустить анимацию при проходе указателя мыши посетителя над анимируемым элементом.

В CSS3 *предоставляется несколько свойств, связанных с анимацией* и позволяющих управлять способом и временем проигрывания анимации (а также сокращенная версия, охватывающая все отдельные свойства). Как минимум, чтобы заставить анимацию выполняться, нужно указать имя, которое было дано исходной анимации (*<имяАнимации>* в правиле `@keyframes`), и продолжительность анимации.

🖫 **Имя анимации: свойство `animation-name`** определяет применяемую к элементу анимацию:

```
animation-name: none | <имяАнимации>
```

✓ `none` — ключевое слово, означающее отсутствие анимации (значение по умолчанию). Также используется, чтобы отменить анимацию элемента из группы элементов, для которых задана анимация;

✓ `<имяАнимации>` — имя анимации, которое связывает правило `@keyframes` с селектором. Если к элементу применяется несколько анимаций, можно использовать одно свойство *`animation-name`*, в котором имена анимаций перечислить через запятую, например

```
animation-name: glow4, moveBlock, moving-vertically;
```

Каждое имя используется для выбора ключевого кадра в правиле `@keyframes`, которое предоставляет значения свойств для анимации. Если отсутствует имя анимации в описанных правилах `@keyframes` или отсутствуют свойства для анимации, анимация не будет выполняться.

🖫 **Продолжительность анимации: свойство `animation-duration`** определяет продолжительность одного цикла анимации. Задается в *секундах* (`s`) или *миллисекундах* (`ms`). Синтаксис

`animation-duration` полностью соответствует свойству `transition-duration` для переходов (см. п. 2.6.2).

🖫 **Временная функция: свойство `animation-timing-function`** описывает, как будет развиваться анимация между каждой парой ключевых кадров. Синтаксис свойства `animation-timing-function` полностью соответствует свойству `transition-timing-function` для переходов (см. п. 2.6.2).

🖫 **Повтор анимации: свойство `animation-iteration-count`** указывает, сколько раз проигрывается цикл анимации:

```
animation-iteration-count: <число>
```

Анимация будет повторяться указанное `<число>` раз. Если `<число>` не является целым числом, анимация закончится в середине последнего цикла. Отрицательные числа недействительны. Значение `0` вызывает мгновенное срабатывание анимации. Начальное значение `1` означает, что анимация будет воспроизводиться от начала до конца один раз.

Варианты записей свойства:

```
animation-iteration-count: infinite;
animation-iteration-count: 3.5;
animation-iteration-count: 2;
animation-iteration-count: 1, 0, infinite;
```

🖫 **Направление анимации: свойство `animation-direction`** определяет, должна ли анимация воспроизводиться в обратном порядке в некоторых или во всех циклах:

```
animation-direction: normal | reverse | alternate |
                     alternate-reverse
```

Принимаемые константные значения:

✓ `normal` — значение по умолчанию: все повторы анимации воспроизводятся так, как они были изначально определены;

✓ `reverse` — все повторы анимации воспроизводятся в обратном направлении от того, как они были изначально определены;

✓ `alternate` — каждый нечетный повтор цикла анимации воспроизводятся в нормальном направлении, каждый четный повтор воспроизводится в обратном направлении;

✓ `alternate-reverse` — каждый нечетный повтор цикла анимации воспроизводятся в обратном направлении, каждый четный повтор воспроизводится в нормальном направлении.

Когда анимация воспроизводится в обратном порядке, временные функции свойства `animation-timing-function` также меняются местами. Например, при воспроизведении в обратном порядке функция `ease-in` будет вести себя как `ease-out`.

🖫 **Задержка анимации: свойство `animation-delay`** определяет, когда анимация начнется. Задается в *секундах* (`s`) или *миллисекундах* (`ms`), значение по умолчанию — `0s`. Аналогично свойству `transition-delay` для переходов (см. п. 2.6.2).

Свойство `animation-delay` определяет длительность задержки между началом анимации (когда анимация применяется к элементу через свойства) и когда она начинает выполняться. *Отрицательные значения разрешены*, такая задержка начинает анимацию с определенного момента внутри ее цикла, т. е. со времени, указанного в задержке. Это позволяет применять анимацию к нескольким элементам со сдвигом фазы, изменяя лишь время задержки. Чтобы анимация началась с середины, нужно задать отрицательную задержку, равную половине времени, установленном в `animation-duration`.

🖫 **Состояние элемента до и после воспроизведения анимации: свойство `animation-fill-mode`** определяет, какие значения применяются анимацией вне времени ее выполнения. Когда анимация завершается, элемент возвращается к своим исходным стилям. По умолчанию анимация не влияет на значения свойств `animation-name` и `animation-delay`, когда анимация применяется к элементу. Кроме того, по умолчанию анимация не влияет на значения свойств `animation-duration` и `animation-iteration-count` после ее завершения. Свойство `animation-fill-mode` может переопределить это поведение:

```
animation-fill-mode: none | forwards |
                     backwards | both |
```

Значения:

✓ `none` — значение по умолчанию: состояние элемента не меняется до или после воспроизведения анимации, т. е. но как только анимация завершится, браузер отобразит элемент в его первоначальном виде;

✓ `forwards` — после того, как анимация завершится, анимируемый элемент сохранит тот вид, который у него был по завершении анимации;

✓ `backwards` — в течение периода, определенного с помощью `animation-delay`, анимация будет применять значения свойств, определенные в ключевом кадре, которые начнут первую итерацию анимации: это либо значения ключевого кадра `from` (при `animation-direction: normal` или `animation-direction: alternate`), либо значения ключевого кадра `to` (при `animation-direction: reverse` или `animation-direction: alternate`).

✓ `both` — позволяет оставлять элемент в первом ключевом кадре до начала анимации (игнорируя положительное значение задержки) и задерживать на последнем кадре до конца последней анимации.

Таким образом, **определение анимации с применением вышеописанных свойств** выглядит следующим образом:

CSS

```
@keyframes fadeOut {
  from { opacity: 1; }   /* элемент видимый */
  to { opacity: 0; }     /* элемент прозрачный */
}

/* применяем анимацию к классу .fade */
.fade {
  animation-name: fadeOut;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
```

CSS

```
11    animation-iteration-count: 2;
12    animation-direction: alternate;
13    animation-delay: 5s;
14    animation-fill-mode: forwards;
15  }
```

Как и в случае с `transition`, в CSS3 есть универсальное свойство `animation`, в котором можно указать сразу все параметры анимации в одной строке, разделяя их пробелом.

🖫🖫 **Краткая запись анимации: свойство `animation`.** Все параметры воспроизведения анимации можно объединить в одном свойстве `animation`, перечислив их через пробел::

```
animation: [ <animation-name> ||
             <animation-duration> ] |
             <animation-timing-function> |
             <animation-delay> |
             <animation-iteration-count> |
             <animation-direction>
```

Для воспроизведения анимации **достаточно указать только два свойства** — `animation-name` и `animation-duration`, остальные свойства примут значения по умолчанию. Порядок перечисления свойств не имеет значения, единственное, время выполнения анимации `animation-duration` обязательно должно стоять перед задержкой `animation-delay`.

Таким образом, записать анимацию к селектору класса `.fade` из предыдущего примера (строки 7–15) можно следующим образом:

CSS

```
1   .fade {
2     animation: fadeOut 2s ease-in-out 2 alternate
3                        5s forwards;
4   }
```

В общем случае можно ограничится записью с двумя обязательными параметрами:

```
.fade { animation: fadeOut 2s; }
```

Если к двум обязательным параметрам нужно добавить третий (к примеру, `animation-iteration-count`), можете сделать следующую запись:

```
.fade { animation: fadeOut 2s 2; }
```

Если нужно применить к элементу сразу несколько анимаций и задать им разные настройки, то каждую группу значений следует отделять запятой:

CSS

```
div {
    animation: firstAnimation 450ms ease-in,
               secondAnimation 1.5s linear alternate;
}
```

Контрольные вопросы для самопроверки знаний

1. Дайте определение каскадным (многоуровневым) таблицам CSS. Как расшифровывается аббревиатура «CSS»?
2. В чем заключается основная цель разработки CSS?
3. Как в рамках каскадных таблиц можно определить понятия «стиль», «таблица стилей» и «каскадирование»?
4. Кто является разработчиком спецификации Cascading Style Sheets?
5. Перечислите методы вывода документа, которые стали возможными благодаря удобству и универсальности каскадных таблиц стилей.
6. Перечислите основные возможности первой версии CSS1.
7. Какими возможностями в дополнение к CSS1, обладала вторая версия каскадных таблиц CSS2?
8. В чем отличие уточненной версии CSS 2.1 (уровень 2, ревизия 1) от базовой версии CSS2?

9. Приведите пример описания стиля CSS. В чем, с позиции синтаксиса, отличие CSS от HTML?
10. Какую функцию выполняет селектор в описании стиля?
11. В чем заключается объявление свойства и как это объявление записывается с позиции синтаксиса CSS?
12. Приведите примеры расширенной и компактной форм записи стилей CSS для одних и тех же селекторов.
13. При подключении нескольких стилевых файлов, по какому принципу браузер расставит приоритеты для переопределения стилей?
14. Какие стили называют стилями автора, пользователя и браузера? В какой очередности эти стили обрабатываются браузером?
15. Какую роль играет ключевое слово !important при объявлении стилевых правил?
16. Какими методами можно добавить стили к разметке HTML? Как распределяются приоритеты применения стилей, добавленных к HTML разметке разными методами?
17. Какие виды селекторов применяются в CSS?
18. Как объявить селектор типов (тегов)? Какое воздействие он оказывает на разметку CSS?
19. Приведите примеры объявления селекторов классов. Какие ограничения накладываются на имена классов?
20. В каких случаях используются ID-селекторы и как они объявляются в коде CSS?
21. Как можно объявить групповые селекторы для определенных html-элементов веб-страницы?
22. Как можно объявить универсальный селектор для выборки всех html-элементов веб-страницы?
23. Что такое контекстные селекторы и к какому элементу из пары контекстных будут применяться стили?
24. Что такое дочерние селекторы и к какому элементу из пары родительский-дочерний будут применяться стили?
25. В каких случаях необходимо использовать дочерние селекторы, а в каких — контекстные селекторы?
26. Что такое смежные (соседние) селекторы? В каких случаях можно использовать смежные селекторы?
27. Перечислите все виды селекторов атрибутов. Приведите примеры применения разных видов селекторов атрибутов.

28. Что такое псевдокласс? Как они объявляются для тега?
29. Перечислите псевдоклассы, определяющие состояния элементов.
30. Перечислите абсолютные и относительные единицы измерения в CSS? В чем их принципиальное различие, и какие единицы измерения применяются чаще остальных?
31. Какие способы задания цвета предоставляет спецификация CSS?
32. Как вы понимаете выражение «безопасный веб-цвет» (browser-safe color)?
33. Какие свойства CSS используются при шрифтовом оформлении веб-страницы?
34. Раскройте понятия «стандартный шрифт», «нестандартный шрифт» и «безопасный шрифт». Чем они отличаются друг от друга?
35. Приведите синтаксис и пример использования собирательного свойства font.
36. Какие свойства используются при оформлении текстов CSS?
37. В чем заключается принцип блочной модели CSS? Какие компоненты включает в себя блочная модель?
38. Какие свойства используются для управления фоном содержимого блока?
39. Какими свойствами можно регулировать размеры и оформление полей, отступов и рамок?
40. Что такое позиционирование? Перечислите и объясните принцип используемых в CSS четырех типов позиционирования.
41. Какие свойства участвуют в выравнивании и обтекании элементов веб-страницы?
42. С помощью каких свойств достигаются преобразования элементов в CSS? Какие виды преобразований доступны для изменения с помощью каскадных таблиц стилей?
43. Как в CSS осуществляются переходы? Что необходимо, чтобы задать переход?
44. Как происходит создание анимации в CSS?
45. Из каких значений и каких свойств анимации состоит синтаксис краткой записи анимации — animation?

Список литературы к главе 2

1. XML // Википедия : свобод. энцикл. — 2020. — URL: **https://ru.wikipedia.org/wiki/XML** (дата обращения 21.01.2021).
2. What is CSS? // Cascading Style Sheets home page. — 2021. — URL: **https://www.w3.org/Style/CSS/** (дата обращения 21.01.2021).
3. CSS4 не будет… потому что он давно прошел. Встречайте «CSS8»! // CSS-live : сайт. — 2020. — URL: **https://css-live.ru/css/css4-ne-budet-potomu-chto-on-davno-proshel-vstrechajte-css8.html** (дата обращения 21.01.2021).
4. Красота CSS дизайна // Сад CSS Дзена: сайт. — 2021. — URL: **http://www.csszengarden.com/tr/ru/** (дата обращения 21.01.2021).
5. Notepad++ // Википедия : свобод. энцикл. — 2021. — URL: **https://ru.wikipedia.org/wiki/Notepad%2B%2B** (дата обращения 21.01.2021).
6. Programmer's text editor // Jedit : сайт. — 2020. — URL: **http://www.jedit.org/** (дата обращения: 21.01.2021).
7. Мержевич, В. Каскадирование / В. Мержевич // HtmlBook.ru: самоучитель CSS. — 2010. — URL: **http://htmlbook.ru/samcss/kaskadirovanie** (дата обращения: 21.01.2021).
8. Мержевич, В. Преимущества стилей / В. Мержевич // HtmlBook.ru: самоучитель CSS. — 2010. — URL: **http://htmlbook.ru/samcss/preimushchestva-stiley** (дата обращения: 21.01.2021).
9. Макфарланд, Д. Большая книга CSS3 / Д. Макфарланд. — 3-е изд. — Санкт-Петербург : Питер, 2014. — 608 с.: ил. — (Серия «Бестселлеры O'Reilly»).
10. Мержевич, В. Соседние селекторы / В. Мержевич // HtmlBook.ru : самоучитель CSS. — 2010. — URL: **http://htmlbook.ru/samcss/sosednie-selektory** (дата обращения: 21.01.2021).
11. Комплексные селекторы // WebReference : продвинутые уроки по HTML и CSS : сайт. — 2020. — URL:

https://webref.ru/layout/advanced-html-css/complex-selectors (дата обращения: 21.01.2021).
12. Единицы измерения: px, em, rem и другие // Javascript.ru : CSS для JavaScript-разработчика. — 2020. — URL: **https://learn.javascript.ru/css-units** (дата обращения: 21.01.2021).
13. Cascading Style Sheets. Level 2. Revision 2 (CSS 2.2). Specification // W3C : First Public Working Draft, 12 April 2016. — 2016. — URL: **https://www.w3.org/TR/CSS22/** (дата обращения: 21.01.2021).
14. CSS Values and Units Module. Level 3 // W3C : Editor's Draft, 8 October 2020. — 2020. — URL: **https://drafts.csswg.org/css-values-3/** (дата обращения: 21.01.2021).
15. Иваница, С. В. Веб-типографика. Искусство оформления текстов для Интернета / С. В. Иваница. — Донецк : УНИТЕХ, 2013. — 384 с. : ил.
16. Аноприенко, А. Я. Интернет-технологии для студентов и преподавателей : учеб. пособие. Кн. 1 / А. Я. Аноприенко, С. В. Иваница, Т. В. Завадская. — Донецк : УНИТЕХ, 2015. — 260 с. : ил.
17. Список стандартных шрифтов Windows // Fortress-Design. — 2011. — URL: **http://www.fortress-design.com/spisok-standartnyh-shriftov-windows/** (дата обращения: 21.01.2021).
18. Microsoft typography documentation // Microsoft Docs : сайт. — 2021. — URL: **http://www.microsoft.com/typography/fonts/product.aspx** (дата обращения: 21.01.2021).
19. Microsoft typography // Fonts supplied with Mac OS. — 2021. — URL: **http://www.microsoft.com/typography/fonts/mac.htm/** (дата обращения: 21.01.2021).
20. Code Style: Most common fonts for Linux and Unix // Linux and Unix family font survey results. — URL: **http://www.codestyle.org/css/font-family/sampler-UnixResults.shtml** (дата обращения: 21.01.2021).
21. An easy way to install Microsoft's TrueType core fonts on linux // Sourceforge : сайт. — 2021. — URL: **http://corefonts.sourceforge.net/** (дата обращения: 21.01.2021).
22. Border-radius // HtmlBook.ru : сайт. — 2021. — URL: **http://htmlbook.ru/css/border-radius** (дата обращения: 21.01.2021).

23. Корбан, О. Adobe Flash: клиент скорее мертв, чем жив? / О. Корбан // eTutorium : сайт, 28 окт. 2019 г. — 2019. — URL: **https://etutorium.ru/blog/otkaz-ot-adobe-flash** (дата обращения: 21.01.2021).
24. Mathematical Description of Transform Functions // CSS Transforms Module. Level 1. Editor's Draft, 30 October 2020. — 2020. — URL: **https://drafts.csswg.org/css-transforms/#mathematical-description** (дата обращения: 21.01.2021).
25. .This is a block element // The Matrix Construction Set. — 2020. — URL: **http://www.useragentman.com/matrix/** (дата обращения: 21.01.2021).
26. Transform-origin // HtmlBook.ru : сайт. — 2021. — URL: **http://htmlbook.ru/css/transform-origin** (дата обращения: 21.01.2021).
27. CSS Transitions // W3C Working Draft, 11 October 2018. — 2018. — URL: **https://www.w3.org/TR/css-transitions-1/** (дата обращения: 21.01.2021).
28. CSS Easing Animation Tool // Ceaser : сайт. — 2021. — URL: **https://matthewlein.com/tools/ceaser** (дата обращения: 21.01.2021).
29. Animatable CSS properties // MDN Web Docs : сайт — 2020. — URL: **https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties** (дата обращения: 21.01.2021).

ЗАКЛЮЧЕНИЕ

Итак, вторая книга из серии учебных пособий по дисциплине «Интернет-технологии» прочитана до конца! Если вы ее, уважаемый читатель, внимательно прочли и параллельно практиковались на основе изученной информации (надеемся, в большинстве случаев — успешно!), отвечали на контрольные вопросы в конце каждого раздела, просматривали рекомендуемую литературу, то можно вас поздравить — вы максимально эффективно поработали с этой книгой.

Коллектив авторов признателен каждому читателю за изучение изложенного в книге материала, и уверен, что полученная информация и знания помогут каждому в достижении своих целей.

Работая над курсом «Интернет-технологии», авторы ежедневно сталкиваются с все более ускоряющимся развитием мировой IT-сферы: возникновением новых технологий, постоянным усовершенствованием специализированного программного обеспечения, развитием и автоматизацией многих интернет-процессов. Это придает достаточно стабильную мотивацию к развитию читаемого курса путем написания и издания новых учебных и учебно-методических пособий (и планируемых в ближайшее время издания практикума и записи циклов тематических видеолекций). Поэтому именно в эту книгу (напоминаем,

что это вторая книга цикла учебных пособий) были включены проверенные временем и устоявшиеся фундаментальные основы языка гипертекстовой разметки (рассматриваемые в рамках изучения языка разметки html-элементы будут включены в спецификацию версии HTML 5) и более глубокая проработка каскадных таблиц стилей (многие из рассматриваемых стилевых свойств поддерживаются только версией CSS 3).

Данное учебное пособие освещает важные вопросы курса, но охватывает всего часть того, что авторы хотят и могут донести до вас. Остается пожелать вам ознакомиться с первой книгой цикла пособий с общим названием «Интернет-технологии для студентов и преподавателей», а также терпения в ожидании новых работ коллектива авторов в ключе развития учебного и учебно-методического обеспечения дисциплины «Интернет-технологии».

Успехов вам, дорогие читатели!

Портал магистров ДонНТУ (masters.donntu.org)

Русский | English

Обеспечение дисциплины литературой

Фотогалереи магистров

Темы выпускных работ

ДонНТУ

Компьютерное образование в ДонНТУ

Рекомендуемое учебное пособие по курсу:

Веб-типографика. Искусство оформления текстов для Интернета

ДонНТУ инициирован в 2000 году в рамках проекта научно-образовательной сети URAN

Информационный сайт «Одаренное поколение»

История ДонНТУ в одной картине

ОБРАЩЕНИЕ к магистрантам и посетителям портала:

Аноприенко А.Я.,

Портал магистров ДонНТУ

ОБРАЩЕНИЕ РЕКТОРА К МАГИСТРАМ, ВЫПУСКНИКАМ ДОНЕЦКОГО НАЦИОНАЛЬНОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

ПЕРСОНАЛЬНЫЕ САЙТЫ МАГИСТРОВ

Магистратура Донецкого национального технического университета

(для навигации используйте численные значения в таблицах)

Факультеты и институты (с 2010 г.)	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Факультет компьютерных наук и технологий (КНТ)	106	103	105	115	108	24	25	48	80	80
Факультет компьютерных информационных технологий и автоматики (КИТА)	40	39	48	49	43	7	7	52	56	56
Факультет радиотехники и специальной подготовки	8	8	9	6	9	2				
Электротехнический факультет (ЭТФ)	55	60	38	33	33	16	16	64	64	64
Физико-металлургический факультет (ФМФ)	50	47	47	48	35	20	29	29	29	29
Институт горного дела и геологии (ИГГ)	63	63	62	44	48	37	17	17	17	17
Факультет инженерной механики и машиностроения (ИММ)	45	37	29	28	41	21	21	48	48	48
Факультет экологии и химической технологии (ЭХТ)	33	57	44	25	29	12	12	38	38	38
Инженерно-экономический факультет (ИЭФ)	45	75	59	62	64	13	45	45	45	45
Институт информатики и искусственного интеллекта			42							
Итого	445	489	483	410	410	152	172	341	377	

Избранные коллективные фото магистров факультета КНТ

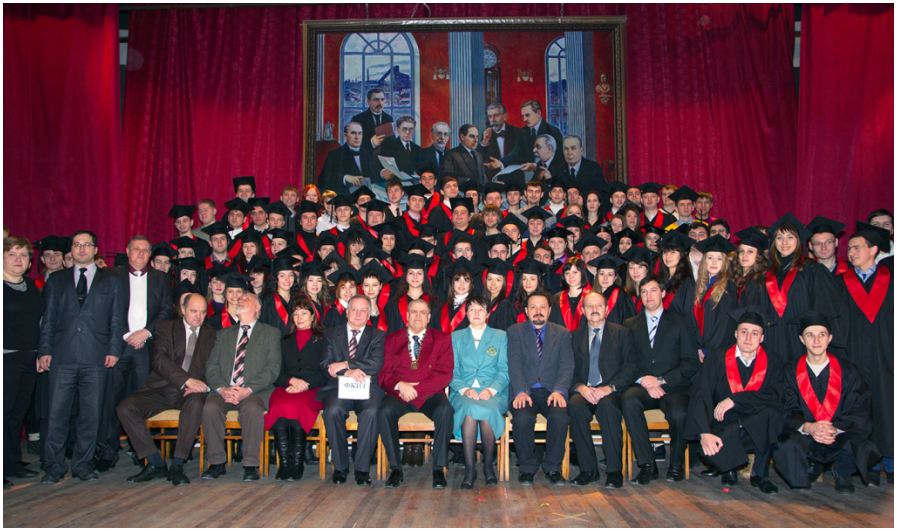
Магистры – 2008

Магистры – 2009

Магистры – 2010

Магистры – 2011

Магистры – 2012



Магистры – 2013

Магистры – 2014

Магистры – 2016

Учебно-методическое обеспечение курса «Интернет-технологии»

А. Я. Аноприенко,
С. В. Иваница,
Т. В. Завадская

Интернет-технологии для студентов и преподавателей

Учебное пособие

Книга написана на основе материалов курса «Интернет-технологии» для магистрантов, читаемого в ДонНТУ с 2000 года.

Эта книга — первая из серии «Интернет-техно-логии для студентов и преподавателей», в которую вошел материал о веб-дизайне, веб-типографике, о работе с графической информацией в Интернете и программном обеспечении для работы с сайтами.

Для широкого круга читателей: старшие школьники и студенты, магистранты и аспиранты, молодые ученые, преподаватели.

Аноприенко, А. Я. Интернет-технологии для студентов и преподавателей: учебно-методическое пособие: книга первая / А. Я. Аноприенко, С. В. Иваница, Т. В. Завадская. — Донецк: ДонНТУ, УНИТЕХ, 2015. — 260 с.: ил.

С. В. Иваница

Веб-типографика. Искусство оформления текстов для Интернета

Учебное пособие

В книге подробно рассмотрены принципы профессиональной работы с текстом, которые накладывают многовековые традиции издательского дела на технику разработки текстового содержимого сайтов в современных условиях. Подробно рассматривается разметка и оформление текстов средствами HTML и CSS.

Овладев правилами и приемами веб-типографики, можно повысить эффективность сайта и, в частности, расширить аудиторию пользователей.

Также внимание уделяется синтаксическому оформлению текста, начальным сведениям о представлении текстовых данных в компьютере, принципам кодирования символов и цветов.

Книга содержит множество полезных советов, наглядных иллюстраций, подробных примеров. Материалы книги структурированы по принципу «от простого — к сложному» и будут одинаково интересны как начинающим веб-разработчикам и веб-дизайнерам, так и профессионалам.

Иваница, С. В. Веб-типографика. Искусство оформления текстов для Интернета. / С. В. Иваница. — Донецк: ДонНТУ, УНИТЕХ, 2013. — 384 с. : ил.

А. Я. Аноприенко,
С. В. Иваница,
К. А. Сидоров

Методические указания по лабораторным работам по дисциплине «Интернет-технологии»

Методические указания

Целью лабораторного курса «Интернет-технологии» является поэтапное выполнение основных работ, связанных с созданием персонального тематического сайта, основное содержание которого посвящено теме выпускной работы магистра.

Методические указания содержат материалы по лабораторным работам, примеры их выполнения и контрольные вопросы.

Для студентов уровня профессионального образования «магистр» всех направлений подготовки и форм обучения.

Методические указания к лабораторным занятиям по дисциплине «Интернет-технологии» [Электронный ресурс] : для студентов уровня профессионального образования «магистр» всех направлений подготовки и форм обучения / ГОУВПО «ДОННТУ», фак. КНТ ; каф. компьютерной инженерии ; сост. А. Я. Аноприенко, С. В. Иваница, К. А. Сидоров – Электрон. дан. (1 файл: 1,94 Мб). – Донецк: ДОННТУ, 2020. – Систем. требования: Acrobat Reader.

А. Я. Аноприенко,
С. В. Иваница,
К. А. Сидоров

Методические указания к самостоятельной работе по дисциплине «Интернет-технологии»

Методические указания

Целью практической части курса «Интернет-технологии» является создание персонального тематического сайта, основное содержание которого посвящено теме выпускной работы магистра.

Методические указания к выполнению самостоятельной работы являются руководством по разработке сайта, основанным на рассмотрении соответствующего шаблона, содержащего примеры реализации отдельных разделов сайта.

Для студентов уровня профессионального образования «магистр» всех направлений подготовки и форм обучения.

Методические указания к самостоятельной работе по дисциплине «Интернет-технологии» [Электронный ресурс] : для студентов уровня профессионального образования «магистр» всех направлений подготовки и форм обучения / ГОУВПО «ДОННТУ», фак. КНТ, каф. компьютерной инженерии ; сост. А. Я. Аноприенко, С. В. Иваница, К. А. Сидоров – Электрон. дан. (1 файл: 2,65 Мб). – Донецк: ДОННТУ, 2020. – Систем. требования: Acrobat Reader.

Для заметок

Для заметок

Учебное издание

АНОПРИЕНКО Александр Яковлевич
ИВАНИЦА Сергей Васильевич

ИНТЕРНЕТ-ТЕХНОЛОГИИ ДЛЯ СТУДЕНТОВ И ПРЕПОДАВАТЕЛЕЙ

Книга вторая

ISBN 978-966-8248-95-5

Редакционно-техническое оформление,
компьютерная верстка: С. В. Иваница
Дизайн обложки: С. В. Иваница

Подписано к печати с готового оригинал-макета 21.03.2021.
Формат 60х90 1/16. Бумага мелованная.
Гарнитура «Times New Roman». Печать — лазерная.
Уч.-изд. л. 8,73. Ус. печ. л. 15,41.
Заказ № 2103. Тираж 350 экз.

Отпечатано в типографии
«Цифровая типография»
Адрес: г. Донецк, ул. Артема, 138 а
Тел.: +380 (62) 388-07-30

АНОПРИЕНКО АЛЕКСАНДР ЯКОВЛЕВИЧ

Профессор кафедры «Компьютерная инженерия» Донецкого национального технического университета (ДонНТУ, г. Донецк).

Автор курса «Интернет-технологии» и основатель портала магистров ДонНТУ (masters.donntu.org). Читает лекции по курсу «Интернет-технологии» с 2000 года.

ИВАНИЦА СЕРГЕЙ ВАСИЛЬЕВИЧ

Доцент кафедры «Компьютерная инженерия» Донецкого национального технического университета. Автор учебного пособия «Веб-типографика. Искусство оформления текстов для Интернета». С 2010 года ведет практические занятия, а с 2014 года — читает лекции по курсу «Интернет-технологии».

Книга написана на основе материалов курса «Интернет-технологии» для магистрантов, читаемого в Донецком национальном техническом университете с 2000 года.

Эта книга — вторая из книг серии «Интернет-технологии для студентов и преподавателей», в которую вошел материал о развитии гипертекстовых технологий и появившегося на их основе языка гипертекстовой разметки. Также читателям предложен обширный материал о каскадных таблицах стилей, при изучении которых детально рассматриваются основные принципы и подходы к созданию динамических элементов веб-страниц.

Издание приурочено к 100-летию Донецкого национального технического университета

Сайт ДонНТУ: **donntu.org**

Портал магистров ДонНТУ: **masters.donntu.org**

ДЛЯ ШИРОКОГО КРУГА ЧИТАТЕЛЕЙ:
СТАРШИЕ ШКОЛЬНИКИ И СТУДЕНТЫ, МАГИСТРЫ И АСПИРАНТЫ,
МОЛОДЫЕ УЧЕНЫЕ, ПРЕПОДАВАТЕЛИ ВСЕХ КВАЛИФИКАЦИОННЫХ УРОВНЕЙ