

Владимир Дронов



JavaScript и AJAX в Web-дизайне

2-е издание



- HTML, CSS, JavaScript и AJAX
- Web-сценарии и события
- Управление Web-обозревателем и содержимым Web-страницы
- Графика и анимация
- Web-формы и базы данных
- Фильтры и преобразования

Наиболее
полное
руководство

В ПОДЛИННИКЕ®

УДК 681.3.06
ББК 32.973.26-018.2
Д75

Дронов В. А.

Д75 JavaScript и AJAX в Web-дизайне: 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2008. — 736 с.: ил. — (В подлиннике)

ISBN 978-5-9775-0251-1

В книге описывается все, что должен знать Web-дизайнер: принципы создания Web-страниц, язык JavaScript, основы написания Web-сценариев, работа с содержимым Web-страницы, обработка данных, введенных в Web-форму, особенности различных Web-обозревателей, использование баз данных, фильтров и преобразований, графика, анимация и пр. Изложение сопровождается большим количеством подробно разобранных примеров и полезных советов. Особое внимание уделено вопросам совместимости Web-сценариев с различными Web-обозревателями.

Второе издание книги, ранее вышедшей под названием "JavaScript в Web-дизайне", полностью переработано и дополнено с учетом современных технологий, дан вводный курс AJAX.

Для Web-дизайнеров

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Екатерина Капалыгина</i>
Компьютерная верстка	<i>Натальи Караваевой</i>
Корректор	<i>Виктория Пиотровская</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.06.08.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 59,36.

Тираж 2500 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0251-1

© Дронов В. А., 2008
© Оформление, издательство "БХВ-Петербург", 2008

Оглавление

Введение	1
О чем эта книга?	1
Какие программы будут использоваться в этой книге?	2
Типографские соглашения	3
Благодарности	4
ЧАСТЬ I. ВВЕДЕНИЕ В WEB-ДИЗАЙН И WEB-ПРОГРАММИРОВАНИЕ	5
Глава 1. Что такое Интернет и как он работает	7
Основные принципы работы Интернета	7
Что такое Интернет	7
Сервисы Интернета	9
Клиенты и серверы	9
Протоколы	12
Интернет-адреса.....	15
Основные понятия WWW	17
Web-страницы и Web-сайты	18
Web-обозреватели.....	20
Web-серверы	22
Что дальше?	23
Глава 2. Язык HTML. Создание Web-страниц	24
Введение в язык HTML	25
Основные понятия HTML.....	25
Вложенность тегов	27
Две секции Web-страницы.....	29
Работа с текстом.....	30
Форматирование фрагментов текста	30

Форматирование абзацев	32
Создание списков	34
Управление переносом строк	37
Специальные символы	38
Текст фиксированного формата	40
Работа с гиперссылками	42
Создание гиперссылок	42
Интернет-адреса в WWW	44
Почтовые гиперссылки	46
Якоря	46
Работа с графикой	48
Внедренные элементы	48
Форматы интернет-графики	49
Вставка графических изображений	50
Специальные изображения	51
Работа с таблицами	54
Создание таблиц	54
Название и секции таблицы	57
Объединение ячеек таблиц	58
Реализация всплывающих подсказок	61
Служебные теги HTML	62
Теги каркаса	63
Название Web-страницы	63
Задание кодировки страницы	64
Пролог	66
Комментарии	67
Фреймы	68
Что такое фреймы	68
Создание набора фреймов	70
Использование цели гиперссылки для указания фрейма	73
Дополнительные возможности фреймов и наборов фреймов	74
Будущее HTML	75
Что дальше?	76
Глава 3. Язык CSS. Каскадные таблицы стилей	77
Введение в каскадные таблицы стилей	77
Создание таблиц стилей	78
Разновидности стилей	79
Разновидности таблиц стилей	80
Правила каскадности и приоритет стилей	82
Атрибуты стилей CSS	84

Параметры шрифта.....	85
Параметры фона	88
Параметры абзаца.....	90
Параметры размеров и размещения.....	93
Параметры отступов.....	94
Параметры рамки.....	95
Параметры списков	96
Параметры курсора.....	97
Псевдостили.....	98
Контейнеры.....	100
Физическое и логическое форматирование.....	102
Что дальше?	103
Глава 4. Язык JavaScript	104
Введение в JavaScript	104
Основные понятия JavaScript	104
Типы данных JavaScript	106
Переменные	108
Именованние переменных	109
Объявление переменных.....	109
Операторы.....	110
Арифметические операторы.....	110
Оператор объединения строк	111
Двоичные операторы.....	111
Операторы присваивания.....	112
Операторы сравнения.....	113
Логические операторы	114
Оператор получения типа <i>typeof</i>	115
Совместимость и преобразование типов данных	115
Приоритет операторов	116
Сложные выражения JavaScript	118
Блоки.....	119
Условные выражения	119
Условный оператор ?	121
Выражения выбора	121
Циклы.....	123
Функции	126
Объявление функций.....	127
Функции и переменные. Локальные переменные	128
Вызов функций	129

Присваивание функций. Функциональный тип данных.....	130
Рекурсия.....	130
Встроенные функции JavaScript.....	131
Массивы	134
Ссылки.....	135
Объекты.....	136
Понятия объекта и экземпляра объекта	137
Работа с объектами и их экземплярами.....	137
Объект <i>Object</i> и использование его экземпляров	139
Новые возможности JavaScript, применяемые при работе с объектами	140
Встроенные объекты JavaScript	141
Пользовательские объекты	159
Комментарии	165
Правила написания выражений	166
Что дальше?	167

ЧАСТЬ II. БАЗОВЫЕ ПРИЕМЫ

JAVASCRIPT-ПРОГРАММИРОВАНИЯ.....169

Глава 5. Общие принципы написания Web-сценариев.....	171
Как пишутся Web-сценарии	171
Внутреннее представление страницы. Document Object Model (DOM).....	174
Именованние элементов страницы.....	176
Получение доступа к элементу страницы.....	177
Прямой доступ по имени	178
Доступ через коллекции.....	178
Доступ с помощью свойств и методов DOM.....	180
Особенности работы с таблицами	184
Средства DOM для получения параметров элемента страницы.....	188
Файлы сценариев.....	190
Что дальше?	192
Глава 6. Обработка событий.....	193
События и обработчики событий	194
Обработка событий по модели Internet Explorer	195
Обработка событий по модели Firefox.....	200
Получение дополнительной информации о событии	203
Получение информации о событии в Internet Explorer и Opera	203
Получение информации о событии в Firefox.....	206

Всплытие событий	209
Перехват событий в дочерних элементах в модели обработки событий Firefox	212
Поведение по умолчанию и его отмена	214
Что дальше?	216
Глава 7. Работа с Web-обозревателем.....	217
Получение сведений о Web-обозревателе	217
Работа с окнами Web-обозревателя.....	225
Управление размерами и местоположением окна.....	225
Прокрутка содержимого окна	227
Создание нового окна.....	230
Работа с программно созданными окнами.....	232
Переключение между окнами	233
Закрытие окна	234
Прочие манипуляции с окнами	235
События объекта <i>Window</i>	235
Работа с интернет-адресом текущей страницы	239
Работа с историей Web-обозревателя.....	242
Получение сведений о видеоподсистеме клиентского компьютера	244
Доступ к содержимому фреймов	246
Что дальше?	249
Глава 8. Управление содержимым Web-страницы.....	250
Работа с содержимым страницы.....	251
Изменение названия страницы	251
Изменение содержимого страницы	251
Работа с атрибутами тегов.....	265
Прямой доступ к атрибутам через свойства	265
Использование коллекции <i>attributes</i>	267
Использование методов DOM	269
Работа со стилями	271
События элементов страницы и их обработка	276
События мыши.....	276
События клавиатуры	282
Прочие события	287
Прочие свойства и методы элементов страницы	288
Что дальше?	290
Глава 9. Управление графикой и мультимедийными элементами	291
Работа с обычными графическими изображениями	292

Свойства и события объекта <i>HTMLImageElement</i>	292
Горячее изображение	293
Полоса навигации	295
Предзагрузка графических изображений	301
Работа с картами-изображениями	303
Работа с мультимедийными данными	306
Поддержка мультимедийных данных	307
Модули расширения Web-обозревателя	308
Элементы ActiveX	311
Компромиссное решение: модель расширения + элемент ActiveX	315
Дополнительные параметры	316
Управление элементами ActiveX из сценариев	322
Что дальше?	335
Глава 10. Управление свободно позиционируемыми элементами.	
Анимация на Web-страницах	336
Свободно позиционируемые элементы	336
Что такое свободно позиционируемый элемент	337
Создание свободно позиционируемых элементов	338
Управление свободно позиционируемыми элементами из сценариев	345
Анимация на Web-страницах	350
Простейшая анимация	350
Анимация реального времени	352
Анимация по ключевым точкам	361
Drag'n'drop	368
Что дальше?	377
Глава 11. Работа с данными	378
Вывод данных	378
Вывод данных в строке статуса	379
Вывод данных в окнах-сообщениях	380
Ввод данных	380
Сохранение данных на клиентском компьютере	382
Передача данных между страницами	390
Обработка данных с использованием регулярных выражений	394
Введение в регулярные выражения	395
Средства JavaScript для работы с регулярными выражениями	400
Что дальше?	406
Глава 12. Работа с Web-формами	407
Создание Web-форм и элементов управления	408

Как работают Web-формы	408
Создание Web-форм	411
Создание элементов управления	412
Примеры Web-форм и страниц, получающих данные от Web-форм	428
Работа с Web-формами и элементами управления из сценариев	434
Работа с Web-формами	434
Работа с элементами управления	438
Примеры Web-форм, управляемых сценариями	454
Что дальше?	459

ЧАСТЬ III. ИСПОЛЬЗОВАНИЕ СПЕЦИФИЧЕСКИХ ВОЗМОЖНОСТЕЙ INTERNET EXPLORER И FIREFOX.... 461

Глава 13. Взаимодействие с посетителем (Internet Explorer и Firefox)..... 463

Работа с произвольными фрагментами текста	464
Работа с фрагментом текста в Internet Explorer	464
Работа с фрагментом текста в Firefox	472
Работа с выделенным текстом	482
Работа с выделенным текстом в Internet Explorer	482
Работа с выделенным текстом в Firefox	484
Работа с Буфером обмена (Internet Explorer)	489
Реализация drag'n'drop с переносом данных (Internet Explorer)	493
Использование диалоговых окон HTML (Internet Explorer)	503
Модальные диалоговые окна HTML	504
Немодальные диалоговые окна HTML	510
HTML-приложения (Internet Explorer)	514
Что дальше?	520

Глава 14. Работа с базами данных (Internet Explorer) 521

Введение в базы данных	521
Что такое база данных	521
Текстовая база данных	523
Реализация работы с базами данных	524
Загрузка базы данных	525
Привязка элементов страницы к данным	527
Программная привязка элементов страницы к данным	530
Средства управления TDC из сценариев	534
Фильтрация и сортировка записей средствами TDC	539

Что дальше?	542
Глава 15. Фильтры и преобразования (Internet Explorer)	543
Фильтры	543
Создание фильтров	543
Программное управление фильтрами	551
Преобразования	555
Создание преобразований	555
Программное управление преобразованиями	562
Применение преобразований к странице	564
Что дальше?	565
Глава 16. Поведения и HTML-компоненты (Internet Explorer)	566
Поведения	566
Создание простых поведений	567
Подключение поведений к элементам страницы	570
Специфические события поведений и их обработка	571
Создание свойств поведения	573
Создание методов поведения	582
Создание событий поведения	584
Программное управление поведением	587
Стандартные поведения Internet Explorer	588
HTML-компоненты	591
Создание HTML-компонентов	591
Использование HTML-компонентов	596
Дополнительные параметры HTML-компонента	598
Программное управление HTML-компонентами	599
Что дальше?	599
Глава 17. Рисование на Web-странице (Firefox)	600
Канва	601
Контекст рисования	602
Рисование простейших фигур	602
Задание цвета, уровня прозрачности и толщины линий	603
Рисование сложных фигур	605
Как рисуются сложные контуры	605
Перо. Перемещение пера	606
Прямые линии	607
Дуги	607
Кривые Безье	608

Прямоугольники	611
Задание стиля линий.....	612
Использование сложных цветов	614
Линейный градиентный цвет.....	614
Радиальный градиентный цвет.....	616
Графический цвет	618
Вывод внешних изображений.....	620
Преобразования системы координат.....	623
Сохранение и загрузка состояния	623
Перемещение начала координат канвы.....	624
Поворот системы координат	625
Изменение масштаба системы координат.....	626
Управление наложением графики	627
Маски.....	629
Что дальше?	629

ЧАСТЬ IV. НАЧАЛА ТЕХНОЛОГИИ AJAX..... 631

Глава 18. Работа с данными XML 633

Язык XML	634
XML DOM.....	636
Вставка данных XML в Web-страницу	638
Простейшая страница, обрабатывающая данные XML	643
Более сложная страница, обрабатывающая данные XML	645
Страница, выводящая данные XML по частям с возможностью листания	648
Что дальше?	651

Глава 19. Асинхронный обмен данными..... 652

Введение в технологию AJAX	652
Реализация асинхронного обмена данными	655
Простая страница, реализующая технологию AJAX.....	660
Загрузка данных в ответ на действия посетителя	663

Заключение..... 669

ПРИЛОЖЕНИЯ 673

Приложение 1. Часто используемые теги и атрибуты HTML, объявленные стандартами как устаревшие	675
--	------------

Устаревшие теги.....	675
Устаревшие атрибуты тегов.....	678
Приложение 2. Специальные символы HTML.....	684
Приложение 3. Коды и обозначения цветов.....	688
Приложение 4. Свободно распространяемые библиотеки для JavaScript-программистов.....	694
JsHttpRequest.....	694
Prototype.....	695
DOJO.....	695
Предметный указатель.....	697



Глава 2

Язык HTML. Создание Web-страниц

Вооружившись необходимыми теоретическими знаниями об Интернете и интернет-технологиях, перейдем к более практическим вещам, а именно — к созданию Web-страниц. Эти страницы мы во второй части книги будем "оживлять" с помощью Web-сценариев.

Итак, что мы уже знаем о Web-страницах из *главы 1*?

Мы знаем, что Web-страницы — суть обычные текстовые файлы, созданные в любом простейшем текстовом редакторе (том же Блокноте) и сохраненные с расширением `htm[1]`.

Мы знаем, что эти файлы содержат текст, который составляет содержимое страницы, и особые команды, называемые тегами и используемые для задания внешнего вида или назначения тех или иных *элементов* Web-страницы. С помощью этих тегов можно, например, оформить фрагмент текста как отдельный абзац, выделить его полужирным шрифтом или курсивом или превратить в заголовок.

Мы знаем, что для создания Web-страниц используется язык HTML. Этот язык определяет набор тегов, их назначение, правила написания и расстановки в тексте страницы.

Язык HTML был разработан в начале 80-х годов прошлого века Тимом Бернерсом-Ли. Кроме самого HTML, этот примечательный человек также создал первые программы Web-обозревателя и Web-сервера — и первые Web-страницы. Можно сказать, что он создал WWW и Интернет в современном его виде. И даже удостоился за это рыцарского звания!

Шло время. HTML пополнялся новыми возможностями. Менялись номера версий этого языка. Самая последняя версия — 4.01 — вышла в конце 90-х годов прошлого века; все современные Web-обозреватели ее поддерживают.

Бернерс-Ли давно отошел от активных дел, передав язык HTML на попечение организации *World Wide Web Consortium* (сокращенно — *WWWC* или, что встречается чаще, *W³C*). Это название дословно можно перевести как "Комитет Всемирной паутины". W³C издает весьма увесистые труды, описывающие набор тегов HTML и необходимые требования к Web-обозревателям.

Введение в язык HTML

Язык HTML лучше всего изучать на примерах. Так что давайте не будем болтать впустую, а сразу же создадим нашу первую Web-страничку.

Основные понятия HTML

Операционная система Windows уже содержит все нужные инструменты. И первый из этих инструментов — текстовый редактор Блокнот. Запустим его и наберем приведенный далее текст (или, как говорят программисты, код) на языке HTML. Выглядит он устрашающе, но настоящего Web-дизайнера (и, тем более, Web-программиста) так просто не испугаешь.

```
<HTML>
  <HEAD>
    <TITLE>Web-страница</TITLE>
  </HEAD>
  <BODY>
    <H1>Пример Web-страницы</H1>
    <P>Это простейшая Web-страничка, созданная в стандартном
    <EM>Блокноте</EM> и отображенная в <EM>Microsoft Internet
    Explorer</EM>.</P>
  </BODY>
</HTML>
```

После этого проверим набранный код HTML на ошибки и сохраним в файле с именем 2.1.htm. Только когда будем вводить имя файла в стандартном окне сохранения, заключим его в кавычки, иначе Блокнот по доброте душевной добавит расширение txt, и наш файл получит имя 2.1.htm.txt.

Теперь откроем полученный файл в Web-обозревателе Internet Explorer, для чего достаточно дважды щелкнуть по нему мышью. Internet Explorer — второй инструмент Web-дизайна, который припасла нам Windows. То, что мы увидим в его окне, показано на рис. 2.1.

Вот мы и создали свою первую, совсем простенькую Web-страничку!

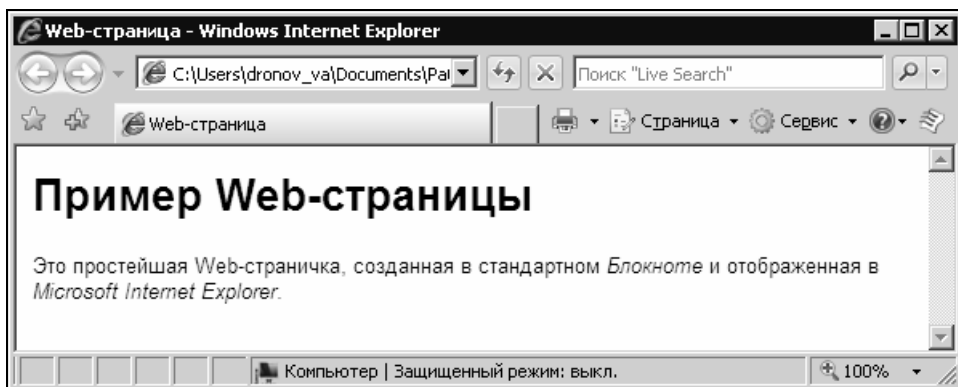


Рис. 2.1. Простейшая Web-страничка

Теперь займемся собственно языком HTML. Снова запустим Блокнот и откроем в нем только что созданную Web-страницу (проще всего это сделать, перетащив файл этой страницы в окно Блокнота). И найдем в ее HTML-коде вот этот фрагмент:

```
<H1>Пример Web-страницы</H1>
<P>Это простейшая Web-страничка, созданная в стандартном
<EM>Блокноте</EM> и отображенная в <EM>Microsoft Internet
Explorer</EM>.</P>
```

Он задает содержимое Web-страницы, видимое в окне Web-обозревателя. Помимо самого текста, здесь присутствуют какие-то слова, заключенные в символы < ("меньше") и > ("больше"). Это и есть теги HTML. Они задают форматирование текста. Скажем, строка "Блокноте" будет выведена курсивом, так как теги и задают курсивное начертание текста. Причем тег помечает начало курсивного фрагмента (*открывающий тег*), а тег — конец (*закрывающий*). А собственно фрагмент текста, заключенный между открывающим и закрывающим тегами, называется *содержимым* тега. Именно к содержимому применяется действие тега.

Также в приведенном фрагменте HTML-кода имеются теги <P> и <H1> (и соответствующие им закрывающие теги </P> и </H1>). Они создают соответственно обычный текстовый абзац и заголовок; при этом Web-обозреватель отобразит их надлежащим образом — см. рис. 2.1.

Еще один полезный и часто употребляемый тег — . Он выделяет текст полужирным шрифтом. Например, если мы изменим HTML-код нашей Web-страницы таким образом:

```
<H1>Пример Web-страницы</H1>
<P>Это простейшая Web-страничка, созданная в стандартном
```

```
<EM>Блокноте</EM> и отображенная в <STRONG>Microsoft Internet Explorer</STRONG>.</P>
```

слова "Microsoft Internet Explorer" отображаются полужирным шрифтом.

Теперь подытожим все, что узнали.

- ❑ Для создания Web-страниц используются особые слова — теги.
- ❑ Тег содержит имя, представляющее собой набор латинских букв и символов < и >, в которые заключается это имя.
- ❑ Закрывающий тег также включает в себя символ /, который помещается между символом < и именем тега.

Традиционно имена тегов набирают прописными буквами. Хотя стандарт HTML этого не предписывает, так HTML-код лучше читается.

Как видно, ничего особо сложного в языке HTML нет. Единственная сложность — это запомнить все нужные теги, но это вопрос времени, опыта и хорошей справочной литературы.

Вложенность тегов

Если внимательно посмотреть на HTML-код нашей страницы, можно заметить, что одни теги вложены в другие. В частности, тег вложен в тег <P>. Такая *вложенность* тегов в HTML — обычное дело.

Когда Web-обозреватель встречает тег, вложенный в другой тег, он складывает эффект от применения этих тегов. Так, в нашем случае слово "Microsoft Internet Explorer" будет частью абзаца и отобразится курсивным шрифтом. То есть Web-обозреватель сложит эффект от применения тегов <P> и .

Если мы заключим в тег , скажем, тег (как мы уже знаем, он задает полужирное начертание текста), то содержимое последнего будет выведено полужирным курсивом. Давайте так и сделаем. Измененный фрагмент нашей Web-страницы будет выглядеть так:

```
<P>Это простейшая Web-страничка, созданная в стандартном  
<EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</STRONG>  
Internet Explorer</EM>.</P>
```

Сохраним полученный файл под именем 2.2.htm и откроем его в Web-обозревателе. Слово "Microsoft" будет выведено полужирным шрифтом, как показано на рис. 2.2.

Обратим внимание на порядок, в котором следуют открывающие и закрывающие теги. Порядок следования закрывающих тегов должен быть обратным тому, в котором следуют открывающие теги. Говоря иначе, теги со всем

их содержимым должны полностью вкладываться в другие теги, не оставляя "хвостов" снаружи.

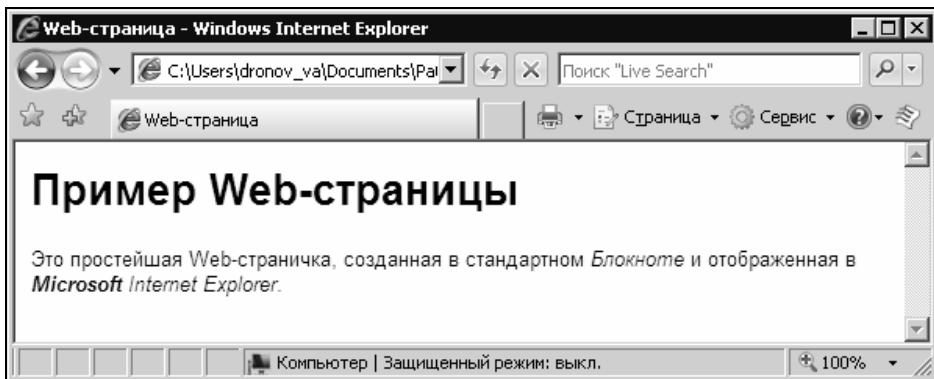


Рис. 2.2. Демонстрация эффекта вложенности тегов HTML

Если же мы нарушим это правило и напишем такой HTML-код (обратите внимание на специально перепутанный порядок следования закрывающих тегов `` и ``):

```
<P>Это простейшая Web-страничка, созданная в стандартном
<EM>Блокноте</EM> и отображенная в <STRONG><EM>Microsoft</STRONG>
Internet Explorer</EM>.</P>
```

то Web-обозреватель может и не отобразить наше творение правильно (хотя Internet Explorer славится своим умением исправлять мелкие ошибки Web-дизайнера). Так что об этом желательно не забывать.

Осталось только выучить несколько новых терминов. Тег, в который вложен данный тег, называется *родительским*. В свою очередь, тег, вложенный в данный тег, называется *дочерним*. Так, для тега `` в приведенном ранее примере тег `<P>` — родительский, а тег `` — дочерний. Любой тег может иметь сколько угодно дочерних тегов, но только один родительский (что, впрочем, понятно — не может же он быть вложен одновременно в два тега).

Уровень вложенности того или иного тега показывает количество тегов, в которые он последовательно вложен. Так, если принять за точку отсчета тег `<P>`, то тег `` будет иметь первый уровень вложенности, так как он вложен непосредственно в тег `<P>`. Тег `` же будет иметь второй уровень вложенности, так как он вложен в тег ``, а тот, в свою очередь, — в тег `<P>`. В сложных Web-страницах уровень вложенности иных тегов может составлять несколько десятков.

Кстати, зачастую HTML-код набирается такой лесенкой, где уровень вложенности тегов показывается величиной отступа слева. Такой код лучше читается, и в нем сразу видны некоторые ошибки (пропущенный закрывающий тег и пр.).

Две секции Web-страницы

Теперь давайте рассмотрим еще несколько тегов, используемых для служебных целей и не отображаемых Web-обозревателем. Они так и называются — *невидимые* теги. (Все остальные уже рассмотренные нами теги были *видимыми*.)

Посмотрим снова на HTML-код нашей Web-страницы. Мысленно удалим из него фрагмент, "отвечающий" за видимое содержимое. Получится вот что:

```
<HTML>
  <HEAD>
    <TITLE>Web-страница</TITLE>
  </HEAD>
  <BODY>
    . . .
  </BODY>
</HTML>
```

Видно, что все теги, задающие содержимое Web-страницы, помещаются внутри парного тега `<BODY>`. Этот тег используется для выделения так называемой *секции тела* Web-страницы. Все видимое содержимое страницы должно находиться в секции тела, то есть в теге `<BODY>`.

Другой парный тег — `<HEAD>` — задает *секцию заголовка* (не путайте с обычным текстовым заголовком, задаваемым тегом `<H1>`). В секции заголовка помещается служебная информация, описывающая саму Web-страницу и используемая Web-обозревателем для внутренних нужд. Среди этой служебной информации может присутствовать *название* Web-страницы, показываемое в заголовке окна Web-обозревателя; оно задается парным тегом `<TITLE>`. Собственно, секция заголовка нашей Web-страницы только ее название — "Web-страница" — и содержит.

И секция заголовка, и секция тега страницы находятся внутри парного тега `<HTML>`. Этот тег располагается на высшем (нулевом) уровне вложенности и не имеет родителя.

Такую структуру оформления HTML-кода страницы — с секциями заголовка и тела и тегам `<HTML>`, `<HEAD>` и `<BODY>` — предписывает стандарт HTML.

Лучше всегда ему следовать, иначе возможны проблемы с некоторыми Web-обозревателями.

Итак, первый шаг в изучении языка HTML мы сделали. Пора продолжить учебу: изучить новые теги и новые возможности, которые они предлагают.

Работа с текстом

Текст — это самое главное во многих Web-страницах. Именно ради текста в большинстве случаев они и создаются. Немногочисленные исключения — например, фотогалереи, — не в счет.

Форматирование фрагментов текста

Наше знакомство с HTML продолжится с рассмотрения тегов, предназначенных для форматирования фрагментов текста, а именно — выделения их особым начертанием.

Мы уже знаем, что парный тег `` выделяет фрагмент текста, являющийся его содержимым, полужирным шрифтом. Аналогично, парный тег `` выделяет фрагмент текста, являющийся его содержимым, курсивом.

Страница открыта в `Microsoft` `Internet Explorer`
Здесь слово "Microsoft" будет выделено полужирным шрифтом, а слова "Internet Explorer" — курсивом.

Вкладывая тег `` в тег `` (или наоборот), можно выделить фрагмент текста полужирным курсивом:

```
<STRONG><EM>Это полужирный курсив</EM></STRONG>
```

Не забываем только о порядке следования закрывающих тегов — он должен быть противоположным порядку открывающих тегов.

Теги `` и `` имеют еще одно назначение. Они позволяют задать степень важности указанных фрагментов относительно "обычного" текста, причем тег `` задает бóльшую важность, нежели тег ``. Так, если мы вернемся к приведенному ранее примеру

Страница открыта в `Microsoft` `Internet Explorer`
то слово "Microsoft" будет помечено как очень важное, а слова "Internet Explorer" — просто важные.

Зачем это нужно? Ну, хотя бы, затем, чтобы выделить слова, на которые посетитель страницы должен обратить внимание. Обычный Web-обозреватель выделит их полужирным шрифтом или курсивом (в зависимости от использованного нами тега), а программы чтения с экрана — скажем, интонацией.

Но что делать, если мы хотим выделить какой-либо фрагмент текста полужирным шрифтом или курсивом, не делая его при этом важным? Следует использовать стили CSS, о которых речь пойдет в *главе 3*. А мы вернемся к тегам HTML.

Следующие два тега, которые мы должны рассмотреть, — это `<SUB>` и `<SUP>`. Оба они парные. Первый выводит свое содержимое нижним индексом, а второй — верхним. Например:

```
H<SUB>2</SUB>O
```

```
E=mc<SUP>2</SUP>
```

В первом случае мы получим на "выходе" H_2O , а во втором — $E=mc^2$.

Стандарт HTML 4.01 определяет еще несколько тегов форматирования фрагментов текста, которые также дают этим фрагментам особый смысл, но используются не столь часто, как `` и ``. Все они перечислены в табл. 2.1.

Таблица 2.1. Прочие теги форматирования фрагментов текста

Тег	Смысл, даваемый им содержимому	Визуальное выделение содержимого
<code><ABBR></code>	Аббревиатура	Подчеркивание точечной линией (не всегда)
<code><ACRONYM></code>	Аналогичен <code><ABBR></code>	Не выделяется
<code><ADDRESS></code>	Адрес "обычной" почты	Курсив
<code><CITE></code>	Цитата	Курсив
<code><CODE></code>	Фрагмент исходного кода программы	Моноширинный шрифт уменьшенного размера
<code></code>	Фрагмент, помеченный для удаления (но не удаленный)	Зачеркивание
<code><DFN></code>	Термин, встречающийся в тексте впервые	Курсив
<code><INS></code>	Фрагмент, вставленный в текст страницы	Подчеркивание
<code><KBD></code>	Текст, который пользователь должен ввести с клавиатуры	Моноширинный шрифт
<code><SAMP></code>	Информация, выведенная какой-либо программой пользователю	Моноширинный шрифт
<code><VAR></code>	Обозначения в тексте имен переменных программы на каком-либо языке программирования	Курсив

Здесь нужно дать некоторые пояснения. Обычно Web-обозреватель выводит содержимое Web-страниц *пропорциональным* шрифтом, в котором символы имеют разную ширину. Такие шрифты используются для вывода текста практически всегда — и другими программами, и самой Windows, — так как набранный ими текст лучше читается. Но некоторые из перечисленных в табл. 2.1 тегов выводят текст *моноширинным* шрифтом, где все символы имеют одинаковую ширину. Моноширинные шрифты традиционно используются для набора исходных текстов программ и для программиста выглядят более привычно, поэтому содержимое тегов `<CODE>`, `<KBD>` и `<SAMP>` выводится именно моноширинным шрифтом.

Вообще, полезность тегов, перечисленных в табл. 2.1, весьма сомнительна. Современные Web-обозреватели не предлагают для них никакой особой поддержки, кроме выделения шрифтом, а этого легко достичь с помощью стилей CSS (см. главу 3). Но они присутствуют в стандарте HTML 4.01, и нам следует о них знать.

Форматирование абзацев

От фрагментов текста перейдем к более крупным "единицам" — абзацам. И посмотрим, что HTML припас нам для их форматирования.

Один тег для форматирования абзацев нам уже знаком. Это парный тег `<P>`, который мы использовали при написании нашей первой Web-страницы. Он, собственно, создает обычный текстовый абзац, не имеющий какого-либо особого значения.

```
<P>Это первый абзац.</P>
```

```
<P>А это второй абзац.</P>
```

При выводе на экран абзаца Web-обозреватели руководствуются следующими правилами:

- содержимое тега `<P>` становится содержимым абзаца;
- если содержимое абзаца не удается вывести в одну строку, выполняется перенос строк по словам;
- по умолчанию содержимое абзаца выравнивается по левому краю;
- от других абзацев он отделяется свободным пространством;
- программы чтения с экрана могут отделять один абзац от другого паузой.

Разбиение текста на абзацы существенно улучшает его читаемость, как и в случае других электронных документов (текстовых, Word и пр.) и обычных, "бумажных" книг. Это понятно.

Если нам нужно поместить в текст страницы заголовок, мы воспользуемся одним из парных тегов `<Hn>`, где n — номер от 1 до 6. Этот номер задает так называемый *уровень* заголовка, степень его "веса" относительно других заголовков и обычных абзацев. Так, уже знакомый нам тег `<H1>` создает заголовок первого уровня — самый "увесистый"; с его помощью создаются заголовки Web-страниц. Тег `<H2>` создает заголовок второго уровня, подходящий для заголовков отдельных частей большого текста, тег `<H3>` — заголовок третьего уровня (заголовки отдельных глав) и т. д.

```
<H1>Заголовок страницы</H1>
<H2>Заголовок первой части</H2>
<H3>Заголовок первой главы</H3>
<P>Первая глава...</P>
<H3>Заголовок второй главы</H3>
<P>Вторая глава</P>
<H2>Заголовок второй части</H2>
<H3>Заголовок третьей главы</H3>
<P>Третья глава...</P>
```

При выводе на экран заголовка правила таковы:

- содержимое тега `<Hn>` становится содержимым заголовка заданного уровня;
- если содержимое заголовка не удастся вывести в одну строку, выполняется перенос строк по словам;
- текст заголовка выводится более крупным шрифтом, нежели содержимое обычного абзаца. Размер шрифта при этом зависит от уровня заголовка;
- по умолчанию содержимое заголовка выравнивается по левому краю;
- от других абзацев он отделяется свободным пространством, величина которого также зависит от уровня заголовка;
- программы чтения с экрана могут выделять заголовок интонацией и более длительной паузой.

Последний тег форматирования абзацев, который мы должны рассмотреть, — это `<BLOCKQUOTE>`. Этот тег применяется, в основном, для оформления больших цитат, вынесенных в отдельный абзац.

```
<BLOCKQUOTE>
☞Писать на книжках очень глупо.
☞Иван Фадееч Кандалупа.
☞ (Саша Черный)
</BLOCKQUOTE>
```

Когда Web-обозреватель встретит этот тег, он примет во внимание вот что:

- содержимое тега `<BLOCKQUOTE>` становится содержимым абзаца, содержащего цитату;
- если содержимое абзаца не удастся вывести в одну строку, выполняется перенос строк по словам;
- по умолчанию содержимое абзаца выравнивается по левому краю;
- от других абзацев он отделяется несколько большим, чем в случае тега `<P>`, свободным пространством;
- абзац выводится с заметным отступом слева и справа;
- программы чтения с экрана могут выделять такой абзац более длительной паузой.

Вот, собственно, и все теги форматирования абзацев.

Нужно сказать еще, что при выводе содержимого любого абзаца — обычного, заголовка или цитаты — Web-обозреватель учитывает следующие дополнительные правила:

- несколько пробелов, следующих друг за другом, трактуются как один пробел;
- переводы строк в HTML-коде трактуются как пробел.

Это позволяет избежать некоторых ошибок нерадивых операторов — тех же двойных пробелов, которые в электронных документах встречаются довольно часто.

Создание списков

Кроме абзацев и заголовков, HTML позволяет создавать списки, содержащие набор пунктов; пункты могут быть помечены либо маркерами, либо нумерацией. В первом случае говорят о *маркированном* списке, а во втором — о *нумерованном*.

Для создания маркированного списка используется парный тег ``. Внутри этого тега помещаются его пункты; каждый из этих пунктов должен помещаться внутри парного тега ``. Например:

```
<UL>
  <LI>HTML;</LI>
  <LI>CSS;<LI>
  <LI>JavaScript.</LI>
</UL>
```

Здесь мы создали маркированный список из трех пунктов. Каждый из этих пунктов по умолчанию помечается кружком с заливкой в качестве маркера.

Нумерованный список создается аналогично, но с помощью другого тега — ``. Вот пример HTML-кода, создающего нумерованный список из трех пунктов:

```
<OL>
  <LI>HTML;</LI>
  <LI>CSS;</LI>
  <LI>JavaScript.</LI>
</OL>
```

Пункты нумерованного списка по умолчанию нумеруются арабскими цифрами. Первый пункт имеет номер 1, второй — 2 и т. д.

А вот еще один пример нумерованного списка:

```
<OL START="3">
  <LI>HTTP;</LI>
  <LI>FTP;</LI>
  <LI>SMTP;</LI>
  <LI>POP3.</LI>
</OL>
```

Он содержит четыре пункта, причем нумерация их начинается с 3. Да, именно так!

Здесь мы столкнулись с так называемыми *атрибутами* — дополнительными параметрами тегов. Тег `` содержит атрибут `START`, указывающий номер, с которого начнется нумерация пунктов списка (*значение* атрибута).

Атрибуты тегов HTML — весьма примечательная вещь, с помощью которой мы можем в некоторых пределах менять обычное поведение тегов. Нужно только помнить следующее:

- атрибуты тегов записываются после имени тега через пробел, прямо внутри символов `<` и `>`;
- значение атрибута записывается после имени атрибута через знак двоеточия и обязательно указывается в кавычках.

Имена атрибутов, как и имена тегов, также набираются прописными буквами, а значения атрибутов — строчными. Хотя стандарт HTML этого не предписывает, так сложилась традиция, да и на читаемости HTML-кода это скажется лучшим образом — все теги и атрибуты видны сразу.

Атрибут `START` может как присутствовать, так и не присутствовать в теге ``; во втором случае Web-обозреватель начнет нумерацию пунктов списка

со значения по умолчанию — единицы (см. пример, приведенный ранее). Говорят, что атрибут `START` *необязательный*.

Тег ``, создающий пункт списка, поддерживает атрибут `VALUE`. Этот необязательный атрибут указывает номер данного пункта; последующие пункты будут нумероваться начиная с заданного номера.

```
<OL>
  <LI>HTTP;</LI>
  <LI>FTP;</LI>
  <LI VALUE="5">SMTP;</LI>
  <LI>POP3.</LI>
</OL>
```

Здесь пункты "HTTP" и "FTP" будут иметь номера 1 и 2, а пункты "SMTP" и "POP3" — 5 и 6, так как для пункта "SMTP" был задан номер 5.

Понятно, что атрибут `VALUE` имеет смысл указывать только в случае нумерованного списка. В случае списка маркированного он будет проигнорирован.

Зачем может понадобиться задание начала нумерации пункта в списке? Например, если нам понадобится "разбить" большой список обычным абзацем, мы можем поступить следующим образом. Сначала мы создадим обычный нумерованный список, содержащий пункты, которые должны следовать до этого абзаца. Потом мы создадим сам абзац и другой список, содержащий пункты, которые должны следовать после абзаца. Напоследок задаем для второго списка начальный номер пункта, следующий за тем, на котором закончилась нумерация первого списка. Так мы обеспечим сквозную нумерацию в обоих списках.

Осталось только сказать, что пункты списков обоих видов выводятся с отступом слева. А промежуток между пунктами делается существенно меньше, чем промежуток между абзацами.

Стандарт HTML 4.01 предусматривает возможность создания списков еще одной разновидности — *списков определений*. Такой список содержит перечень неких терминов, требующих разъяснения, с самими этими разъяснениями. Используются такие списки редко, но нам следует о них знать.

Сам список определений создается парным тегом `<DL>`, внутри которого помещаются термины и разъяснения. Каждый термин помещается в парный тег `<DT>`. Разъяснение этого термина должно находиться сразу же после него в парном теге `<DD>`.

```
<DL>
  <DT>HTTP</DT>
  <DD>Протокол, используемый сервисом WWW</DD>
```

```
<DT>FTP</DT>
<DD>Протокол, используемый сервисом обмена файлами</DD>
</DL>
```

Здесь мы создали список определений, содержащий два термина с разъяснениями.

Пункты списка определений никак не помечаются и не нумеруются. Вообще, стандарт HTML 4.01 отдает оформление такого списка на откуп Web-обозревателям. Большинство из них оформляет их как обычные списки, за тем исключением, что разъяснения выводятся с бóльшим отступом слева, чем термины, а пункты не маркируются и не нумеруются.

Управление переносом строк

Мы уже знаем, что Web-обозреватель сам выполняет перенос строк по словам, если те не помещаются в окно целиком. Стандарт HTML 4.01 предусматривает некоторые средства по управлению процессом переноса строк.

Иногда бывает нужно выполнить перенос строки в заданном месте абзаца. Для этого служит весьма примечательный тег `
`. Примечателен он тем, что является не двойным, как изученные нами теги, а *одинарным*, то есть не имеет закрывающей "пары".

Тег `
` выполняет перенос строки в том месте текста абзаца, где он встретился. При этом перенесенная на другую строку часть текста все равно останется частью абзаца.

```
<P>Сейчас мы выполним перенос строк..<BR>Перенесли!</P>
```

Слово "Перенесли!" в этом абзаце будет перенесено на другую строку.

Тег `
` "работает" в любом случае, даже если Web-обозревателю хватит места, чтобы разместить остаток текста абзаца, расположенного после этого тега, в той же строке. Поэтому достигаемый им эффект называют *жестким переносом строк*.

Рассмотрим другой случай. Мы пишем текст Web-страницы и хотим, чтобы в некоторой фразе перенос строк не выполнялся в любом случае. Такое может случиться, если мы, например, помещаем на Web-страницу фрагмент исходного кода программы на языке программирования, который не допускает переносов строк внутри выражений (о выражениях см. главу 4). (К таким языкам относится, в частности, Microsoft Visual Basic.) Можно ли указать Web-обозревателю не выполнять перенос строк внутри этой фразы?

Можно. И специально для этого предназначен парный тег `<NOBR>`. Помещаем нужную нам "непереносимую" фразу внутри этого тега и наслаждаемся полученным результатом.

```
<P>Эта фраза <NOBR>не будет перенесена на другую строку</NOBR>!</P>
```

ВНИМАНИЕ!

Тег `<NOBR>` не является частью стандарта HTML 4.01 (*нестандартный тег*). Это значит, что некоторые Web-обозреватели его могут не поддерживать. (Хотя самые распространенные все-таки поддерживают.)

Сказав о теге `<NOBR>`, нельзя не упомянуть об одинарном теге `<WBR>`. Этот тег может присутствовать только внутри тега `<NOBR>` и выполняет *мягкий перенос строк*. В месте "непереносимой" строки, где он встретился, Web-обозреватель может выполнить перенос строк, — именно в этом месте и больше ни в каком другом.

```
<P><NOBR>Эта фраза будет перенесена на другую строку только в<WBR>этом  
☞месте.</NOBR></P>
```

Встретив приведенный фрагмент HTML-кода, Web-обозреватель, если понадобится, сможет выполнить перенос строк только между словами "в" и "этом".

В "непереносимых" строках, помещенных в тег `<NOBR>`, можно использовать и тег жесткого переноса `
`. Только нужно ли?..

Специальные символы

Есть, кстати, еще один способ управления переносом строк. Он позволяет запретить перенос строки между определенными словами. Для этого достаточно между данными словами вставить не обычный пробел, а *неразрывный* — он как раз и запрещает Web-обозревателю выполнять перенос строк.

Неразрывный пробел обозначается особой последовательностью символов — *литералом* — ` `. (Обратите внимание: символ точки с запятой в конце обязателен.)

```
<P>Между этими&nbsp;словами запрещен перенос строк.</P>
```

В приведенном примере между словами "этими" и "словами" перенос строк не будет выполняться в любом случае.

Здесь мы столкнулись с так называемыми *специальными символами* HTML. Такие символы не могут быть вставлены в HTML-код напрямую и требуют использования литералов.

Помимо неразрывного пробела, классическим примером специальных символов служат знаки < ("меньше") и > ("больше"). Напрямую мы их вставить в HTML-код никак не сможем — эти символы используются для обозначения тегов и обрабатываются Web-обозревателем особым образом. Так, если мы напишем

```
<P>x > y</P>
```

символ > не будет выведен на экран, да и не факт, что сам абзац будет правильно обработан. Нам придется использовать соответствующий литерал, обозначающий символ > (выделен полужирным шрифтом):

```
<P>x &gt; y</P>
```

Да, символ > обозначается литералом `>`. А символ < — литералом `<`.

В табл. 2.2 представлены наиболее употребляемые специальные символы HTML и соответствующие им литералы.

Таблица 2.2. Некоторые специальные символы HTML и соответствующие им литералы

Специальный символ	Литерал
Кавычка	<code>&quot;</code>
Знак "меньше"	<code>&lt;</code>
Знак "больше"	<code>&gt;</code>
Неразрывный пробел	<code>&nbsp;</code>
Амперсанд	<code>&amp;</code>
Евро	<code>&#8364;</code>
©	<code>&copy;</code>
®	<code>&reg;</code>
Левая кавычка	<code>&#8220;</code>
Правая кавычка	<code>&#8221;</code>
Длинное тире	<code>&#8212;</code>

Заметим, что для вставки символов евро, левой и правой кавычек и длинного тире используется код этих символов. Это своего рода уникальный номер, которым каждый символ представлен в памяти компьютера. Мы еще поговорим о кодах символов в конце этой главы.

Осталось рассмотреть еще один специальный символ HTML, стоящий несколько особняком. Это *горизонтальная линия* — обычная горизонтальная линия, которая может использоваться, чтобы отделить одну часть страницы от другой.

Горизонтальная линия вставляется в страницу с помощью одинарного тега `<HR>` и помещается Web-обозревателем в том месте, где встретился этот тег.

```
<P>Одна часть страницы.</P>
```

```
<HR>
```

```
<P>Другая часть страницы.</P>
```

По умолчанию Web-обозреватель рисует горизонтальную линию в "трехмерном" виде, с ясно видимой тенью. Но если включить в тег `<HR>` особый атрибут `NOSHADE`, тень у линии рисоваться не будет. Этот атрибут интересен тем, что не принимает значения (*атрибут без значения*) — достаточно одного его присутствия в теге.

```
<HR NOSHADE>
```

```
<P>Это линия без тени.</P>
```

Текст фиксированного формата

Рассмотрим еще один способ оформления текста, доступный в HTML. Это последнее, что нам нужно узнать о работе с текстом.

Давайте напишем вот такую страницу:

```
<HTML>
```

```
  <HEAD>
```

```
    <TITLE>Эксперименты с фиксированным форматом</TITLE>
```

```
  </HEAD>
```

```
<BODY>
```

```
  <P>Это
```

```
  небольшой
```

```
  пример
```

```
  Web-страницы.</P>
```

```
  </BODY>
```

```
</HTML>
```

Назовем ее "Эксперименты с фиксированным форматом" (см. содержимое тега `<TITLE>`). Сохраним его в файле `2.3.htm` и откроем в Web-обозревателе. То, что мы увидим, показано на рис. 2.3.

Видно, что содержимое абзаца, разбитое нами на четыре строки, Web-обозреватель вывел в одну. Собственно, он так и должен был поступить, ведь стандарт HTML требует преобразовывать переводы строк в HTML-коде в пробелы и игнорировать несколько следующих подряд пробелов, преобразуя их в один. Web-обозреватель в данном случае делает все правильно.

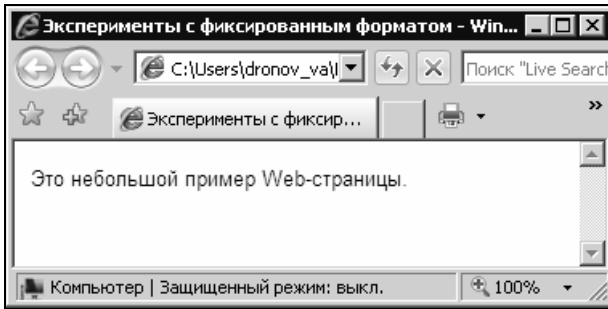


Рис. 2.3. Изначальная страница "Эксперименты с фиксированным форматом"

А теперь мы немного изменим приведенный ранее HTML-код следующим образом (исправления выделены полужирным шрифтом):

```
<HTML>
  <HEAD>
    <TITLE>Эксперименты с фиксированным форматом</TITLE>
  </HEAD>
  <BODY>
    <PRE>Это
небольшой
пример
Web-страницы.</PRE>
  </BODY>
</HTML>
```

Видно, что мы только изменили тег, с помощью которого создается абзац. Сохраним исправленный код в файле 2.4.htm. Если мы откроем новую страницу в Web-обозревателе, то увидим нечто совершенно отличное от виденного ранее (рис. 2.4).

Во-первых, Web-обозреватель вывел текст абзаца "как есть" — со всеми переводами строк. Во-вторых, он вывел этот текст моноширинным, а не пропорциональным шрифтом. Забегая вперед, скажем также, что, если бы мы использовали в тексте несколько следующих друг за другом пробелов, Web-обозреватель также вывел бы их "как есть", без предписанных стандартом HTML преобразований.

А все потому, что мы использовали в странице 2.4.htm тег `<PRE>` вместо тега `<P>`! Этот парный тег предписывает Web-обозревателю вывести заключенный в нем текст без всяких преобразований. То есть создать *текст фиксированного формата*.