федерация
интернет
образования

Учебно-методический комплекс
«Интернет-технологии — образованию»

Начала сайтостроения

***Рекомендовано Министерством образования Российской Федерации
в качестве учебного пособия для системы
дополнительного профессионального образования***

МОСКВА 2002

ББК 32.81
М45

Под редакцией:

Светланы Михайловны Авдеевой,
Павла Юрьевича Белкина,
Александра Александровича Елизарова,
Екатерины Владимировны Алексеевой

Рецензенты:

Никитин Эдуард Михайлович, д-р пед. наук, проф., ректор АПКиПРО;
Рябов Виктор Васильевич, д-р ист. наук, проф., ректор МГПУ,
академик Международной академии информационных процессов
и технологий

Мейксон П.Г., Подбельский В.В.

М45 Учебно-методический комплекс «Интернет-технологии — образованию»: Начала сайтостроения: Учеб. пособие для системы доп. проф. образования. — М.: Федерация Интернет Образования, 2002. — 52 с.

ISBN 5-901891-02-3.

Пособие предназначено для начального ознакомления с основными средствами сайтостроения: языком HTML, каскадными стилями (CSS), скриптами, Java-аплетами. Приведены общие сведения об XML и технологиях формирования WEB-страниц на стороне WEB-сервера.

Пособие рекомендуется для использования в учебных центрах Федерации Интернет Образования и учреждениях дополнительного профессионального образования.

ББК 32.81

ISBN 5-901891-02-3

Содержание

Введение

Сегодня создать Web-страницу, которую потом можно сделать доступной для пользователей Интернет, совсем не сложно. Для этого существуют многочисленные HTML-редакторы, графические редакторы и другие инструментальные средства, научиться работать с которыми достаточно просто. Даже такой широко распространенный текстовый редактор, как Microsoft Word, позволяет автоматически формировать HTML-документы. Однако HTML-страницы, создаваемые с помощью инструментальных средств, не всегда удовлетворяют требованиям искушенного автора. С другой стороны, неподготовленный автор не может понять автоматически сформированный текст исходного описания HTML-страницы и испытывает большие затруднения при необходимости внести в этот текст простейшие изменения или дополнения. Именно такому начинающему автору Web-страниц адресовано данное пособие.

В соответствии с замыслом первого из авторов пособие содержит не учебный, а обзорный материал. Это позволяет читателю получить общее представление о многообразии средств, которые используют современные Web-мастера при разработке Web-документов. Знания (даже хорошего) языка HTML и владения средствами автоматизированной подготовки Web-страниц в настоящее время недостаточно для проектирования конкурентоспособных Web-сайтов. Нужно уметь использовать листы каскадных стилей; программировать скрипты (сценарии), выполняемые как на ЭВМ Web-сервера, так и на ЭВМ Web-клиента; понимать принципы обработки событий, происходящих при общении пользователя с Web-браузером; уметь индексировать создаваемые HTML-документы (например, с помощью мета-тегов); понимать механизмы формирования HTML-кодов «на лету»; знать основные принципы и особенности организации доступа из Web-системы к ресурсам на ЭВМ Web-сервера (к файловой системе, к базам данных); понимать принципиальные отличия Java-аплетов от скриптов на языке JavaScript и т. д.

Практически все перечисленные темы в той или иной мере затронуты в данном пособии, но его объем ни в коем случае недостаточен для подробного их изучения. Предполагается, что читатель получит пособие в составе полного комплекта учебно-методических материалов по программе обучения Федерации Интернет Образования. Задача настоящего пособия — дать общую картину Web-технологий и познакомить читателя с принципами, особенностями и терминами каждой из них. Наибольшее внимание уделено языку HTML, так как он является основным и в настоящее время центральным механизмом, на который «навешиваются» средства наиболее перспективных технологий Web-проектирования.

Несколько слов о соавторстве при написании данного пособия. Общий замысел книги и выбор материала принадлежат П.Г. Мейксону. Его эмоционально написанный текст был переработан и дополнен В.В. Подбельским. Цель переработки состояла в устранении терминологических неточностей и несоответствий в других пособиях упомянутого комплекта учебно-методических материалов.

История языка HTML

Язык разметки гипертекста — HTML (Hypertext Markup Language) — возник на стыке нескольких направлений исследований и разработок. Существуют три фактора, повлиявших на особенности и возможности HTML.

Во-первых, HTML предназначен в основном для создания гипертекстовых документов, и поэтому понимание того, что такое гипертекст, абсолютно необходимо для изучения языка HTML. Первые концепции и разработки, посвященные гипертексту, принадлежали Ванневару Бушу, предложившему концепцию системы Memex в сороковые годы, а также Дагласу Энгельбарту и Теодору Нельсону, работавшим над этой технологией в шестидесятые годы.

Ванневар Буш (1890—1974), научный советник президента Ф. Д. Рузвельта, считается первым, кто дал описание гипертекста. Он сделал это в своей статье 1945 г. «Как мы можем думать» (As We May Think. Atlantic Monthly, July 1945, pp. 101—108, **http://www.isg.sfu.ca/~duchier/misc/vbush/),** в которой призвал не пожалеть в послевоенное время усилий для механизации поиска данных в научной литературе. Он описал браузер — диалоговую машину для просмотра обширной тексто-графической системы и пополнения ее записями.

Эта система, получившая название Memex, включала в себя очень большую библиотеку, а также личные записи, фотографии, зарисовки. Она имела несколько экранов и позволяла вводить помеченную связь между любыми двумя точками библиотеки. Хотя Буш продемонстрировал замечательный дар предвидения, он не разглядел будущей силы компьютера — его Memex использует микрофильмы и фотоэлементы. Однако Буш верно предвидел информационный взрыв и в обоснование своих идей ссылался на потребность в более естественных типах указателей, обеспечивающих информационный поиск.

Термин «гипертекст» предложил Тед Нельсон в 1965 г. Вот как звучит определение гипертекста, которое дал Нельсон в 1987 г.: «...форма письма, которое ветвится или осуществляется по запросу». Иначе говоря, HTML — это «нелинейное письмо», которое «больше, чем текст» (hypertext).

Во-вторых, HTML — это язык разметки, построенный на принципах SGML. В 1986 г. Международная организация по стандартизации (ISO) приняла стандарт ISO-8879, озаглавленный Standard Generalized Markup Language (SGML) — Стандартный обобщенный язык разметки. Стандарт в Интернет доступен по адресу: **http://www.Webreference.com/dlab/books/html/3-0.html**. Названный стандарт вводит SGML — обобщенный *метаязык*, позволяющий строить системы логической, структурной разметки текстов любых разновидностей. При этом управляющие коды, вносимые в текст при такой разметке, не несут никакой информации о внешнем виде документа, а лишь указывают границы и соподчинение его составных частей, т. е. задают его логическую *структуру*. Вот до некоторой степени такой системой и является HTML.

Третьим фактором, определившим специфику HTML, было его применение в Интернете. В 1989 г., после экспериментов с собственными новыми сетевыми технологиями, Европейская лаборатория физики элементарных частиц (CERN) подсоединилась к Интернет. Этот исследовательский центр, расположенный в Женеве (Швейцария), сыграл существенную роль в разработке и принятии стандартов связи и распределенных вычислений, в частности, двух фундаментальных протоколов, которые являются базовыми для Интернет: Transmission Control Protocol (TCP) и Internet Protocol (IP).

В том же 1989 г. Тим Бернерс-Ли предложил глобальную гипертекстовую систему, основанную на более раннем проекте «Enquire» и позволившую соединять связями не только текст, но и графику, звуки, видео. Глобальность этой системы, в отличие, например, от системы Теда Нельсона, состояла в том, что данные распределены по всему миру, а ее основой стал Интернет.

Бернерс-Ли был по образованию физиком. В 1984 г. он пришел в CERN, где занимался разработкой многозадачных операционных систем и компьютерных приложений. Изначальная цель создания новой системы, как и всего Интернет, состояла в обеспечении ученых, прежде всего физиков, новыми, удобными средствами общения, обеспечивающими обмен результатами экспериментов и публикациями.

Через год Бернерс-Ли написал первое клиент-серверное программное обеспечение (гипертекстовую систему Enquire) для того, что теперь известно как World Wide Web (WWW или W3). Эта система была впервые задействована в декабре 1990 г. и дорабатывалась вплоть до 1993 г. За это время был предложен гипертекстовый протокол передачи данных (HTTP—Hypertext Transfer Protocol).

Язык HTML постоянно развивался с момента его создания. Стандартизацией этого языка занимается консорциум World Wide Web, сокращенно W3C (**http://www.w3.org/**). 14 января 1997 г. W3C опубликовал спецификацию языка HTML версии 3.2. Менее чем через год, 18 декабря 1997 г., W3C опубликовал спецификацию HTML 4.0, в которой была окончательно рекомендована к применению концепция CSS.

Простейший HTML-документ

Теперь обратимся к самому языку разметки гипертекста — HTML.

Рассмотрим структуру простейшего HTML-документа. Подготовим в любом текстовом редакторе файл test_1.html, содержащий следующий текст:

```
<html>
<head>
<title>test</title>
</head>
<body>
</body>
</html>
```

Перед нами текст простейшей HTML-страницы или, иначе, Web-страницы. Структурой и форматированием HTML-документа управляют директивы языка HTML, называемые *тегами.* Теги всегда ограничены угловыми скобками. Практически все теги являются парными, т. е. открывающему тегу соответствует закрывающий, который пишется с чертой (/) после открывающей скобки, т. е. его начало имеет вид «</». Каждый тег вводит элемент HTML-документа, обозначаемый именем тега без угловых скобок.

В случае нашего простейшего примера документ начинается с открывающего тега **<html>** и заканчивается закрывающим тегом **</html>**. Они очерчивают границы документа, т. е. вводят элемент с именем **html**, представляющий HTML-документ в целом.

Внутри документа, ограниченного парой **<html> </html>**, первым элементом является **head** — «заголовок документа», ограниченный тегами **<head>** и **</head>**. В этой части документа (в элементе **head**) может содержаться множество важной информации, но, во-первых, содержимое элемента **head** не отображается в окне браузера, а во-вторых, эта информация не обязательная. Единственным исключением является обязательный элемент **title**. Текст, содержащийся между тегами **<title>...</title>**, отображается в заголовке окна браузера как название документа (рис. 1).

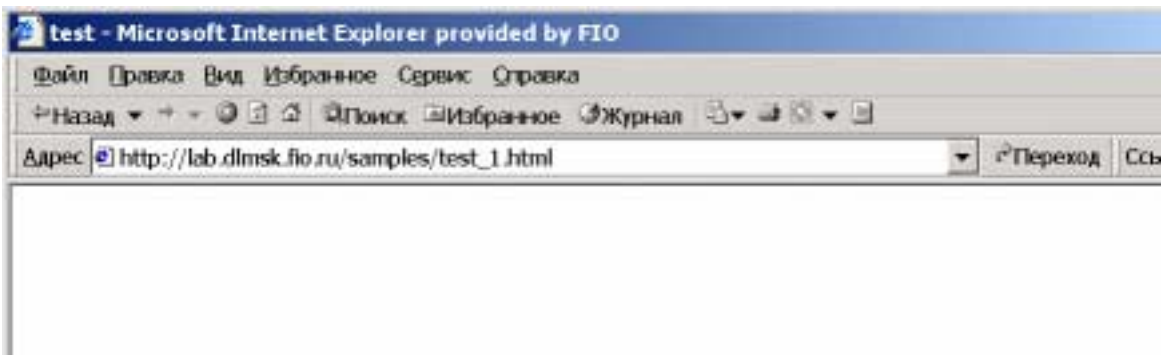

Рис. 1. Часть окна браузера с заголовком и адресной строкой

За заголовком следует элемент **body** — «тело» документа, ограниченное тегами **<body>…</body>**. Здесь и располагается содержимое страницы, которое видит пользователь в рабочем поле окна браузера.

Что касается имен тегов и атрибутов (о них позже) языка HTML, то их можно записывать символами верхнего и нижнего регистров, т. е. как строчными, так и прописными буквами — браузер не делает различия между «большими» и «малыми» буквами. Об этой особенности принято говорить, что HTML является регистронечувствительным языком. Вы можете, например, написать **<TITLE>**, или **<Title>**, или **<title>**, или **<tItLE>**. Кроме того, для браузера не имеет значения расположение тегов на строках документа, важен только их порядок. К примеру, предыдущая запись эквивалентна следующей:

```
<html><head><title>test</title></head><body></body></html>
```

Для обеспечения преемственности при переходе от HTML к более современным яыкам разметки (XHTML, XML) рекомендуется для записи тегов и их атрибутов использовать «малые буквы». В любом случае структуру HTML-документа можно представлять как вложение его элементов (рис. 2).

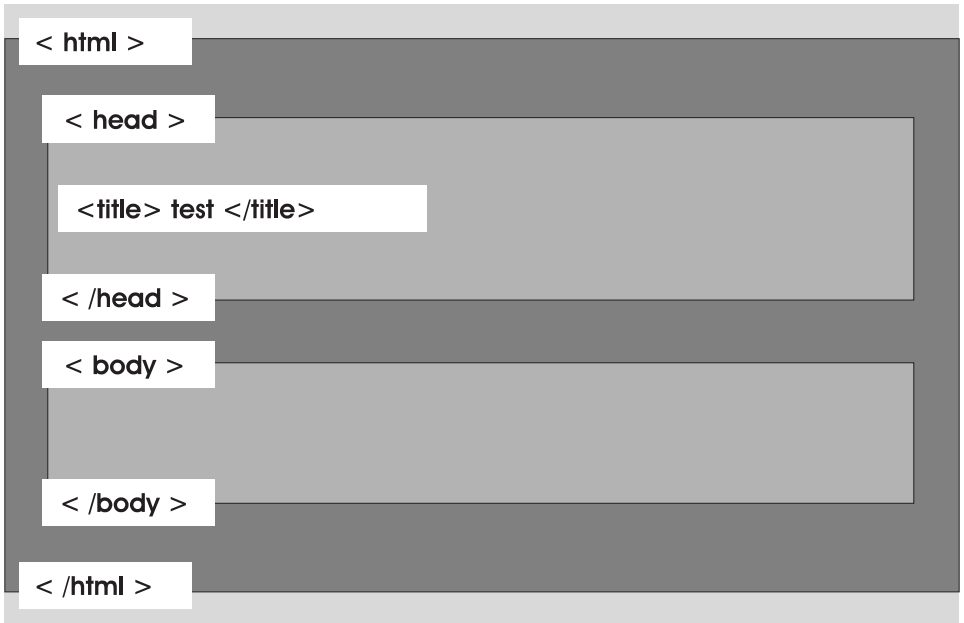

Рис. 2. Вложение элементов простейшего HTML-документа

Перейдем теперь к *содержимому* элемента **body**. Рассмотрим следующий пример:

```
<html>
<head>
<title>пример 2</title>
</head>
<body>
<p>первый абзац текста первый абзац текста
```

```
первый абзац текста первый абзац текста
первый абзац текста первый абзац текста </p>
<p>второй абзац текста второй абзац текста
второй абзац текста второй абзац текста</p>
</body>
</html>
```

В этом примере появился тег ** ** — тег разбиения текста на абзацы. Если не заканчивать абзац закрывающим тегом ** **, то ошибки не возникнет, но общие правила требуют закрывающего тега.

Если набрать этот пример в любом текстовом редакторе и сохранить в файле с расширением htm или html, а затем открыть этот файл в браузере, то окно браузера будет иметь вид, показанный на рис. 3. Здесь в заголовке окна браузера «пример 2» — содержимое элемента **title** и фирменное название браузера «Microsoft Internet Explorer»; в адресной строке — полное имя файла; в рабочем поле окна браузера — текст абзацев из элемента **body**.

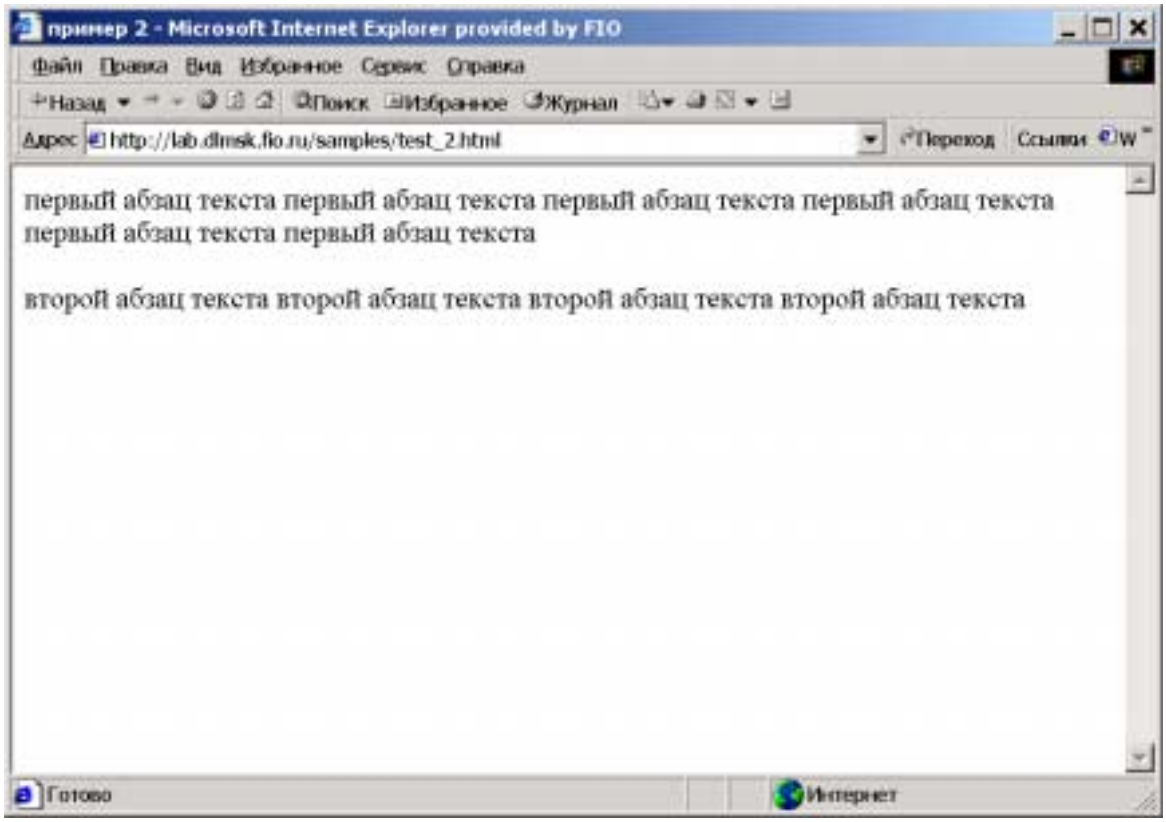

Рис. 3. Отображение документа «пример 2» браузером

При использовании обычного набора тегов абзацы в отображении HTML-документа не имеют абзацного отступа и отделяются в рабочем поле окна браузера друг от друга пустой строкой. Если нужно начать текст с новой строки, не создавая нового абзаца (не добавляя пустую строку), используйте тег **
** — переход на новую строку. Это один из «непарных» тегов — он не имеет закрывающего тега.

Гиперссылки

Рассмотрим теперь практически самое важное свойство HTML — возможность создавать *гиперссылки*. Допустим для примера, что мы хотим создать ссылку на сайт с адресом: **http://www.freeware.ru/**. Раз уж мы употребили термин «сайт», укажем, что Web-сайт — это совокупность связанных одной темой Web-страниц, сохраняемая обычно на одном Web-сервере. Для создания HTML-документа со ссылкой на указанный сайт мы можем написать такой текст:

```
<html>
<head>
<title>коллекция бесплатных программ</title>
</head>
<body>
<p> Одна из лучших <a href="http://www.freeware.ru/"> коллекций
</a> бесплатных программ </p>
</body>
</html>
```

Сохраним приведенный код в файле test_3.html. Если открыть этот файл в браузере, то окно браузера будет иметь вид, показанный на рис. 4.

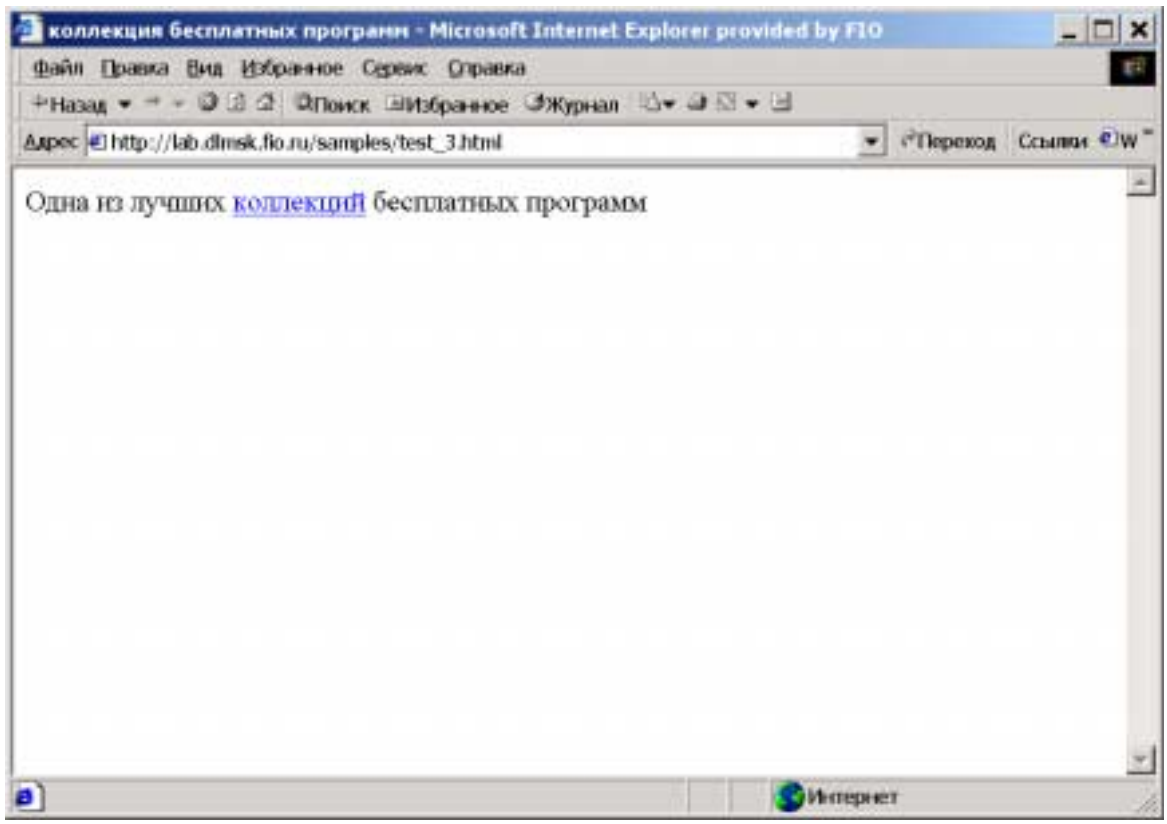

Рис. 4. Отображение примера с гиперссылкой

В отображенном тексте слово «коллекций» является ссылкой и обычно выделяется в рабочем поле подчеркиванием и цветом. Гиперссылка в HTML-документе начинается тегом **<a>**. Здесь мы впервые встречаем не просто тег, а тег с атрибутом. У тега **<a>** могут быть разные атрибуты. Значение атрибута **href** — адрес, по которому выполняется переход при активации ссылки.

В приведенном примере значение атрибута **href** есть полный адрес интернет-ресурса:

```
<a href="http://www.freeware.ru/">
```

Гиперссылка может указывать не на ресурс в Интернет, а на документ на том же Web-сервере:

```
<a href="news/pc_news.htm">новости</a>
```

Здесь гиперссылка «новости» указывает на документ pc_news.htm, находящийся в папке news (размещение папки явно не указано). Такие гиперссылки называются относительными, так как указывают не абсолютный адрес ресурса, а относительный, полностью определенный только в пределах данного HTML-документа или конкретного Web-сайта.

Кроме того, гиперссылка может указывать на другое место в том же документе. То место, на которое нужно сослаться, необходимо пометить с помощью элемента **a** с атрибутом **name: <a name="x1">**, где x1 можно заменить на любое удобное обозначение. А ссылка на метку x1 записывается так:

```
<a href="#x1">текст ссылки</a>
```

Гиперссылка может указывать не только на HTML-документ, но и на любой ресурс и, например, на адрес электронной почты.

В предыдущих примерах ссылки были текстовыми, но ссылкой может быть и графическое изображение. Прежде чем описать, как сделать такую ссылку, покажем, как в HTML-документ вставляется изображение.

Пусть в том же каталоге, где размещен HTML-документ, находится файл с изображением, имеющий имя class.jpg. Включение его в документ будет выглядеть так:

```
<html>
<head>
<title>изображение</title>
</head>
<img src="class.jpg" width="180" height="136" alt="Учебная аудитория">
</body>
</html>
```

Сохраним данный код в файле test_4.html. В браузере мы увидим отображение, показанное на рис. 5.

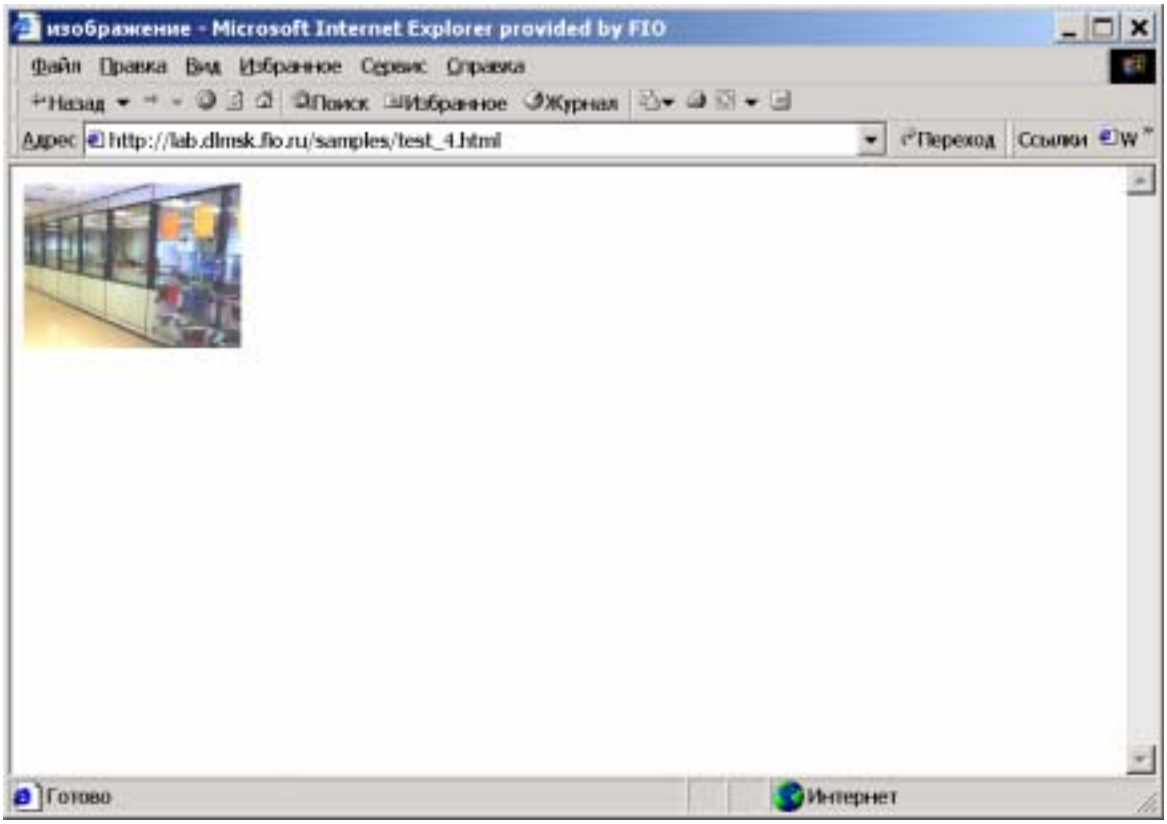

Рис. 5. HTML-страница с изображением

Тег **<img>** в нашем примере имеет следующие атрибуты: **src="class.jpg"** и указывает на файл с изображением, а **width="180"** и **height="136"** задают размеры в пикселах изображения в рабочем поле браузера.

Очень важен атрибут **alt**. В нашем примере он имеет вид: **alt="Учебная аудитория"**. Он важен по следующей причине. Если посетитель страницы хочет ускорить ее загрузку и его интересуют не изображения, а только текст, он может выключить в браузере загрузку изображений. Тогда вместо изображения он увидит текст, указанный как значение атрибута alt (альтернатива изображению). Это особенно важно, если изображение является гиперссылкой.

Пусть требуется, чтобы изображение в предыдущем примере стало гиперссылкой на документ из файла **class.htm**, расположенного в той же папке, что и наш документ. Соответствующий HTML-текст, точнее, элемент **a**, определяющий гиперссылку, выглядит так:

```
<a href="class.htm">
<img src="class.gif" width="180" height="136" border="0"
alt="Учебная аудитория">
</a>
```

В данном примере объединены возможности двух элементов — определения гиперссылки **…** и вставки изображения **<img src="…"…>**.

Начинающие часто делают ошибку, размещая HTML-документы, изображения и другие части одного сайта в разных независимых папках или используя другую терминологию, в разных каталогах файловой системы. Все документы и изображения сайта рекомендуется размещать в одной папке (в одном каталоге) или во вложенных в нее (рис. 6).

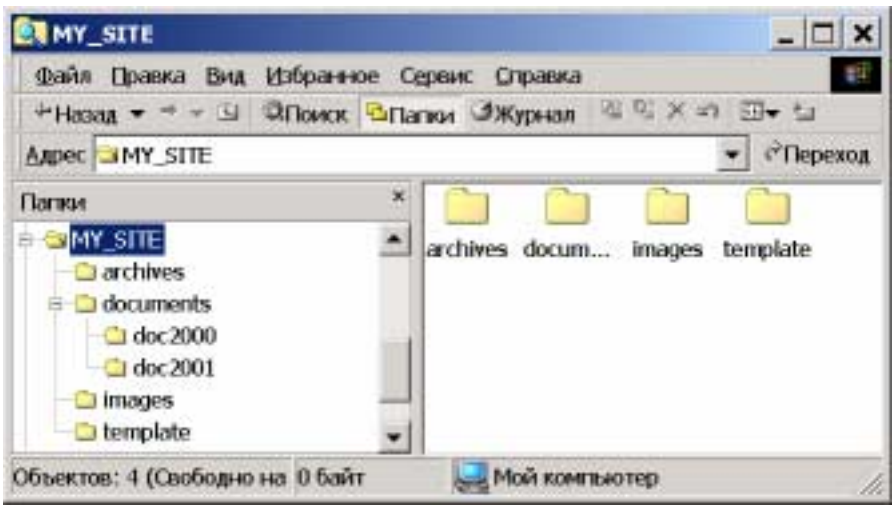

Рис. 6. Рекомендуемое размещение компонентов сайта в каталогах Web-сервера

Кроме того, необходимо при разработке Web-сайта полностью исключать такие «неправильные» ссылки, как в следующем примере:

```
<a href= "file:///D:/Flash/Motion.htm">
неправильная ссылка</a>
```

В этом примере использована абсолютная ссылка на документ Motion.htm, находящийся в папке Flash на локальном диске D. Пока сайт создается на конкретном компьютере и обращения по ссылке выполняются с браузера на том же компьютере, эта ссылка будет работать, но как только сайт будет перенесен на Web-сервер, размещенный на другом компьютере, ссылка прекратит функционировать.

Представление структуры документа

Как мы упоминали выше, язык HTML построен на основе метаязыка Standard Generalized Markup Language (SGML), в котором самым существенным является логическое структурирование документов. К сожалению, в HTML вошло очень немного средств такого структурирования. Основное средство структурирования текстовых документов в HTML — это теги заголовков. Язык HTML предлагает теги **<h1>**, **<h2>**, **<h3>**, **<h4>**, **<h5>**, **<h6>** для заголовков шести уровней. Допустим, что в исходном документе имеются разделы и соответствующие заголовки трех уровней: разделы, главы и параграфы. Таким образом, в нашем примере мы можем использовать такие конструкции:

```
<h1> раздел I </h1>, <h2> глава 1 </h2>, <h3> параграф 1 </h3>
```

Использование этих тегов особенно важно, например, для правильной индексации информационно-поисковыми системами документов, размещенных в Интернете. При отображении документа заголовки выделяются другим, например крупным жирным, шрифтом и размещаются на отдельных строках. Поисковые системы Интернет используют заголовки разделов для распознавания структуры документа. Если Web-дизайнер оформит заголовки не с помощью тегов **<h1> ... <h6>**, а просто выбрав крупный шрифт, размещение на отдельной строке и жирное начертание, то внешне документ будет иметь правильный вид, но для поисковой системы он не будет иметь никакой структуры.

Оформление внешнего вида документа

В HTML существует множество тегов и их атрибутов, позволяющих самым разнообразным способом оформлять внешний вид текста. Например, пусть выражение «оформленный текст» нужно отображать жирным курсивом синего цвета, шрифтовой гарнитурой Arial, размером в 12 пунктов (в типографской системе мер пункт — это наименьшая единица, равная 1/72 дюйма, т. е. 0,376 мм).

Указанным требованиям соответствует следующий HTML-текст:

```
<font face="Arial" size="3">
<i><b><font color="#6600CC">

оформленный текст

</font>
</b></i></font>
```

Здесь первый элемент **font** определяет с помощью своих атрибутов и вложенных элементов, как текст должен отображаться. В данном примере использованы атрибуты: **face="Arial"** — гарнитура шрифта, **size="3"** — размер шрифта. Теги **<i>** и **<b>** определят, соответственно, курсивное и жирное начертание символов (букв). Для задания размера шрифта используются единицы измерения в виде натуральных чисел 1, 2, 3 и т. д. Соответствие между этими единицами и размерами в пунктах в окне браузера может быть различным в разных браузерах. В Internet Explorer используется следующая градация:

Значение атрибута size	1	2	3	4	5	6	7
Пункты (pt)	8	10	12	14	18	24	36

Именно поэтому в нашем примере для шрифта размером в 12pt необходимо выбирать size="3".

Второй тег **<font>** с параметром **color** задает в системе RGB условное обозначение (номер) цвета шрифта. Таким образом, каждый элемент текста, который мы хотим отобразить специальным образом, окружается соответствующими тегами. Объясним правила кодирования цветов, принятые в HTML и использованные в этом примере. Система RGB (Red, Green, Blue) позволяет определять желаемый цвет с помощью трех записанных подряд шестнадцатеричных цифр вида от 00 до FF. Каждое число задает интенсивность соответствующего цветового компонента: R — красного, G — зеленого, B — синего. В начале кода цвета помещается символ #. Если мы захотим каждую букву слова, например «радуга», изобразить отдельным цветом, то получим следующий код:

```
<font color="#FF3300">р</font>
<font color="#00FF00">а</font>
<font color="#FFFF99">д</font>
<font color="#00FFFF">у</font>
<font color="#6666FF">г</font>
<font color="#003399">а</font>
```

Мы не будем здесь более подробно рассматривать оформительские возможности HTML, поскольку для этого существуют справочники, а в Интернет можно найти массу текстов на эту тему, например, по адресу: **http://citforum.ru/internet/html/**

Отметим две характерные черты описанного способа оформления текста.

Размер файла, содержащего сложно оформленный текст, может существенно возрасти за счет большого количества тегов оформления.

При необходимости изменить оформление сайта придется вручную изменять оформление каждого элемента. Это может оказаться довольно трудоемкой работой.

Существует другой подход к оформлению HTML-документов, к рассмотрению которого мы и перейдем.

Каскадные таблицы стилей (CSS)

Мы не рассматриваем здесь полное описание CSS, а даем только его основную концепцию.

Cascading Style Sheets (Каскадные таблицы стилей) — это язык, содержащий набор средств для описания внешнего вида отображения любых HTML-документов. С его помощью можно полностью управлять стилем и расположением каждого элемента Web-страницы, что проще и гораздо функциональнее использования обычного набора HTML-тегов.

Предположим, что текст нужно оформить с помощью CSS так же, как в предыдущем примере, где HTML-код выглядел следующим образом:

```
<font face="Arial" size="3">
<i><b><font color="#6600CC">
оформленный текст
</font>
</b></i></font>
```

Определить стиль можно с помощью специального элемента **style**. Определение стиля для нашего примера выглядит следующим образом:

```
<style type="text/css">
<!--
.mystyle { font-family: Arial, Helvetica, sans-serif;
font-size: 12pt; font-style: italic;
font-weight: bold; color: #6600CC }
-->
</style>
```

Начальный тег определения стиля содержит атрибут **type=«text/css»**, значение которого указывает браузеру, что текст элемента на языке CSS. Определение стиля помещается в скобки HTML-комментария: **«<!- - ...- - >»**. В начале следует «селектор стиля» — идентификатор с точкой перед ним **«.mystyle»**. В фигурных скобках перечисляются свойства данного стиля и их конкретные значения. Каждое свойство задается парой «имя:значение». До двоеточия — имя свойства, после двоеточия — его значение. Прокомментируем приведенный текст определения стиля.

.mystyle (селектор стиля)

font-family: Arial, Helvetica, sans-serif (семейство шрифтов; лучше указывать не один конкретный шрифт, который может отсутствовать на компьютере пользователя, а семейство, обладающее схожими свойствами).

font-size: 12pt (размер шрифта; в данном случае он указан в абсолютных единицах — пунктах, хотя есть и другие способы указания размера шрифта)

font-style: italic (стиль шрифта — курсив)

font-weight: bold («вес» шрифта — полужирный)

color: #6600CC (цвет шрифта, заданный в системе RGB)

Все приведенное описание стиля помещается в виде элемента **style** в заголовок **<head> …</head>** HTML-документа.

Теперь, если мы хотим применить этот стиль к определенной части текста, например, размещенной в элементе **span**, то создаем такой код:

```
<span class="mystyle">оформленный текст</span>
```

Здесь атрибут **class** вводит для элемента **span** так называемое «имя класса». В документе может быть много элементов, но стиль нашего примера будет «действовать» только на те из них, у которых имя класса совпадает с идентификатором **mystyle**, входящим в селектор стиля.

Полный HTML-код будет выглядеть так:

```
<html>
<head>
<title>пример использования CSS</title>
<style type="text/css">
<!- -
.mystyle {font-family: Arial, Helvetica, sans-serif;
font-size: 12pt; font-style: italic;
font-weight: bold; color: #6600CC }
-->
</style>
</head>
<body>
<span class="mystyle">оформленный текст</span>
</body>
</html>
```

Таким образом, использование CSS позволяет определить все необходимые стили в отдельном элементе **<style type="text/css">**, а затем применять их к нужным элементам документа простым указанием селектора стиля в качестве значения атрибута **class: оформленный текст.** В данном примере не проявились преимущества использования листов стилей перед классическими средствами HTML для оформления текстов. Применение таблиц стилей удобно в тех случаях, когда требуется одинаково оформлять много элементов одной HTML-страницы или всех многочисленных страниц одного Web-сайта. В этом случае при необходимости изменить оформление документа достаточно лишь изменить необходимые свойства стиля в его определении. Тогда все элементы, связанные с этим стилем, автоматически изменятся.

Определение стилей можно размещать не на каждой странице сайта. Применение CSS дает еще одну замечательную возможность — позволяет вводить одну *таблицу стилей* для всего сайта. Для этого можно создать отдельный файл с расширением .css со всеми описаниями стилей. Тогда на конкретной странице сайта можно лишь указать ссылку на этот файл, и все описанные в нем стили станут доступными для элементов страницы. Делается это следующим образом. В элементе **<head> …</head>** HTML-документа помещается ссылка на файл стилей, например, на файл с именем main.css:

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

Сам файл стилей main.css содержит только набор определений стилей, например, его текст может быть таким:

```
.link1 {font-family: Arial, Helvetica, sans-serif; font-size: x-small;
font-style: normal; line-height: normal; font-weight: bold;
font-variant: normal; text-transform: none;
color: #663300; text-decoration: none}
.normal {font-family: Arial, Helvetica, sans-serif; font-size: x-small;
font-style: normal; line-height: normal; font-weight: bolder;
font-variant: normal; text-transform: none;
color: #333333; text-decoration: none}
```

Теперь, чтобы изменить оформление элементов на всех страницах сайта, достаточно изменить определения свойств в файле стилей. Можно создавать разные файлы стилей для разных разделов сайта.

Помимо перечисленных возможностей язык CSS предоставляет значительно более широкий, по сравнению со стандартными тегами HTML, спектр средств оформления. Полное описание языка CSS (версия 2) доступно по адресу: **http://www.w3.org/TR/REC-CSS2**. Кроме того, современные визуальные редакторы HTML-документов позволяют включать в документы таблицы стилей с помощью системы меню стилей. На рис. 7 представлено меню определения стилей замечательного HTML-редактора Dreamweaver 4 фирмы Macromedia.

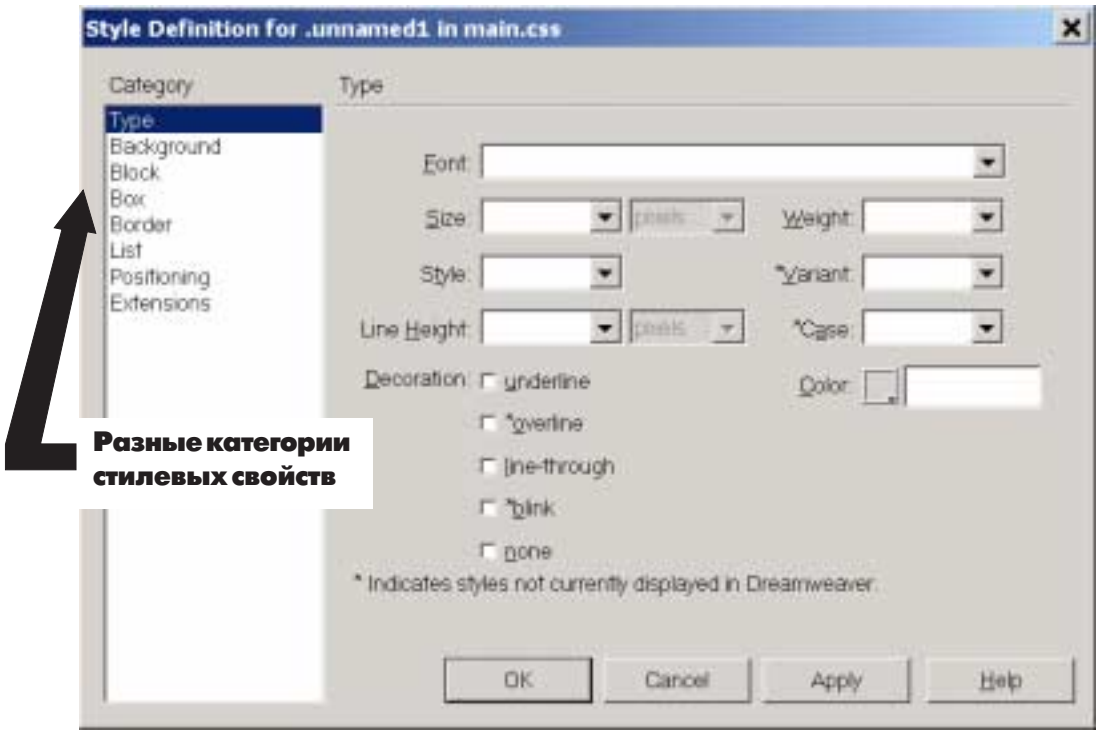

Рис. 7. Панель выбора свойств стилей в редакторе Dreamweaver

Таблицы и слои

Стандартные HTML-теги имеют весьма ограниченные возможности для управления взаимным расположением различных элементов на Web-странице. Web-дизайнеры обычно используют для этих целей таблицы. Рассмотрим кратко правила создания таблиц, а потом их применение для размещения элементов на странице. Допустим, мы создаем таблицу, содержащую 3 столбца и 2 строки. HTML-код будет таким:

```
<table width="100%" border="0" cellspacing="0" cellpadding="0">
```

(Элемент вводит таблицу шириной в 100% по отношению к отображению страницы, толщина рамки 0, расстояние между ячейками 0, отступ содержимого ячейки от ее границ 0. Для точности следует отметить, что при таких значениях атрибутов рамка таблицы и разграничительные линии между ячейками не будут отображаться на экране.)

```
<tr> (начало 1-й строки таблицы)
<td></td> (ячейка 1-го столбца 1-й строки)
<td></td> (ячейка 2-го столбца 1-й строки)
<td></td> (ячейка 3-го столбца 1-й строки)
</tr>
<tr> (начало 2-й строки таблицы)
<td></td> (ячейка 1-го столбца 2-й строки)
<td></td> (ячейка 2-го столбца 2-й строки)
<td></td> (ячейка 3-го столбца 2-й строки)
</tr>
</table> (конец таблицы)
```

В данном примере дана только схема таблицы. Элементы **td** пусты. Именно в них между тегами **<td>** и **</td>** должны размещаться данные, определяющие содержимое ячеек таблицы. В качестве содержимого ячеек таблицы могут использоваться текстовые данные и изображения. Оформлять в табличном виде текстовые данные совсем несложно. Покажем на примере, как используют таблицы не по прямому назначению, т. е. не для размещения в ячейках текстовых данных, а для такого оформления страницы, которое не достигается с помощью стандартных тегов. Предположим, что мы хотим разместить в середине страницы два прилегающих друг к другу рисунка со сдвигом по вертикали (рис. 8).

Соответствующий HTML-код при использовании таблицы для размещения изображений будет выглядеть так:

```
<table cellspacing="0" cellpadding="0" border="0">
<tr>
<td width="151" height="50" valign="top"> </td>
<td width="180" height="50" valign="top"> </td>
<td width="1" height="50" valign="top"> </td>
<td width="154" height="50" valign="top"> </td>
</tr>
<tr>
<td width="151" height="87" valign="top"> </td>
```

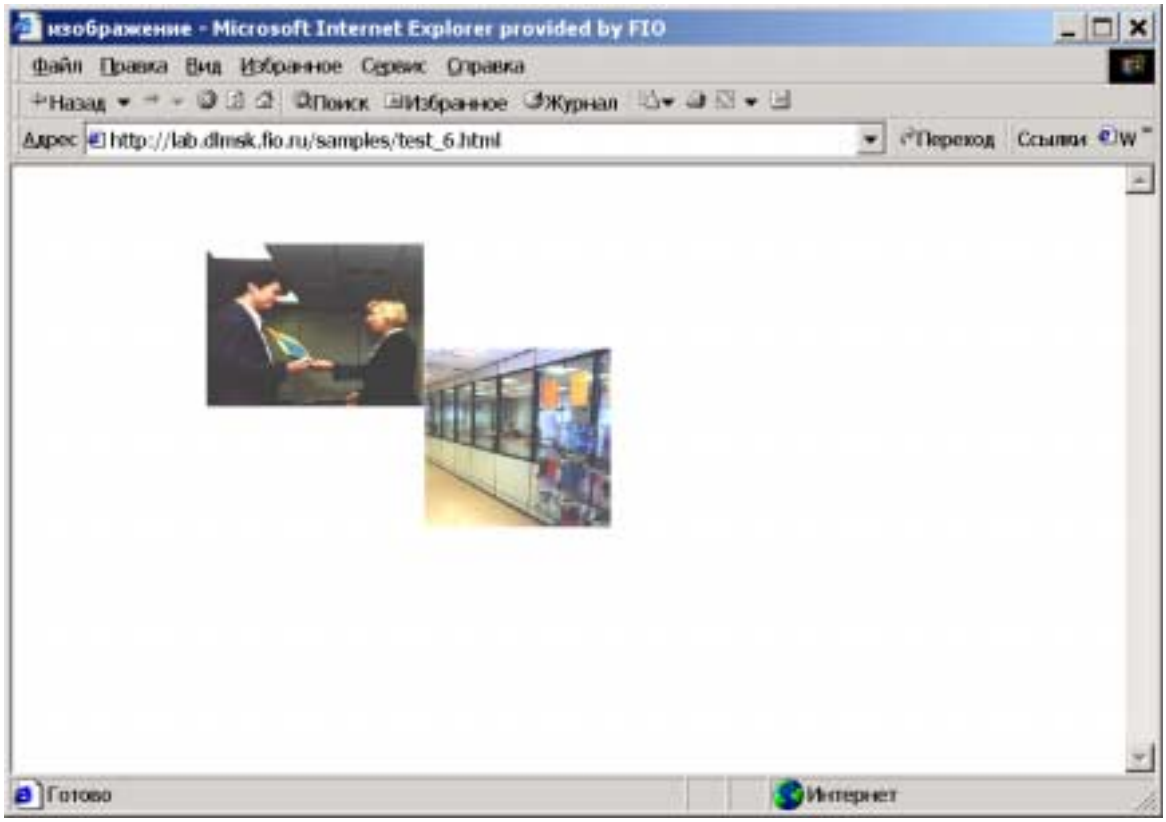

Рис. 8. Желаемое размещение изображений

```
<td width="180" height="235" rowspan="2" valign="top">
<img src=" /bild.jpg" width="180" height="135"></td>
<td width="1" height="87" valign="top"> </td>
<td width="154" height="87" valign="top"> </td>
</tr>
<tr>
<td width="151" height="148" valign="top"> </td>
<td width="1" height="148" valign="top"> </td>
<td width="154" height="148" valign="top">
<img src="class.gif" width="154" height="148"></td>
</tr>
</table>
```

Разбирая приведенное HTML-описание таблицы, смотрите на рис. 9. В таблице 3 строки и 4 колонки ячеек разной величины.

Как в предыдущем примере, атрибуты в теге **<table>** означают: **cellspacing** — расстояние между ячейками; **cellpadding** — отступ содержимого ячейки от ее границ; **border** — толщина рамки таблицы, равная нулю, поэтому рамка изображаться не будет.

Атрибуты в тегах **<td>: width** — ширина ячейки (в пикселах); **height** — высота ячейки (в пикселах); **valign = «top»** — размещение содержимого вверху ячейки.

rowspan = "2" — увеличение высоты за счет объединения двух ячеек смежных строк (текущей и следующей). Именно в эту ячейку под действием элемента

```
<img src = "/bild.jpg" width = "180" height = "135">
```

помещается изображение из файла bild.jpg.

В последней, третьей, строке всего 3 элемента **td**, которые определяют 1-, 3- и 4-ю ячейки таблицы. Вторая ячейка «слилась» со второй ячейкой строки 2, и соответствующий элемент **td** здесь не нужен.

Четвертая ячейка строки 3 включает изображение учебного класса из файла class.gif.

Рис. 9. Размещение изображений в таблице

Мы видим, что для желаемого (см. рис. 8) размещения изображений пришлось использовать довольно сложную таблицу, причем при отображении сделаны невидимыми рамка таблицы и границы ячеек (они условно показаны пунктиром на рис. 9).

Версия языка HTML 4.0 позволяет использовать для позиционирования любых элементов на Web-странице слои, вводимые средствами CSS. Сам слой является прямоугольной областью, контейнером, содержащим другие элементы. Посмотрим, как то же самое расположение изображений можно реализовать с помощью слоев. Разместим «вызовы» тех же самых изображений внутри контейнерных элементов **div**. В качестве атрибутов элементов **div** используем **style** — определяющий стиль отображения в соответствии с правилами CSS:

```
<div style="position:absolute;
left:159px; top:61px; width:180px; height:135px; z-index:1">
<img src="bild.jpg" width="180" height="135">
</div>
<div style="position:absolute;
left:340px; top:148px; width:154px; height:148px; z-index:2">
<img src="class.gif" width="154" height="148">
</div>
```

Основное отличие слоя от других прямоугольных контейнерных HTML-элементов состоит в том, что слои можно перемещать по странице, накладывать друг на друга и делать их видимыми и невидимыми, когда страница уже отображена в рабочем поле браузера. Слой в нашем примере начинается тегом ** **. Его атрибут **style** определяет стилевые свойства слоя. Среди них свойство **position:absolute** (**position** — имя свойства, **absolute** — его значение) означает абсолютное позиционирование слоя. В этом случае положение левого верхнего угла слоя имеет жесткие координаты относительно левого верхнего угла страницы, обозначаемые свойствами: **left:159px; top:61px** (px — обозначение единицы измерения «пиксел»). Далее указаны размеры слоя: **width:180px** — горизонтальный и **height:135px** — вертикальный. Смысл свойства **z-index** мы рассмотрим чуть позже. А дальше следует тег вставки изображения:

```
<img src="bild.jpg" width="180" height="135">
```

Отметим, что в этом теге размеры изображения задаются атрибутами HTML-элемента IMG. При задании значений в пикселах не требуют указания единиц измерения. Принципиально важно, что тег включения изображения **<img>** находится в контейнере слоя. Поэтому можно управлять расположением изображения на странице, изменяя параметры размещения слоя. Таким образом, включив изображение в слой, можно разместить его в произвольном месте страницы. В слой может быть вставлено не только изображение, но практически любой элемент, например текст или таблица.

Использование слоев для позиционирования элементов страницы существенно проще, универсальнее и удобнее использования таблиц. Кроме того, слои дают еще и совершенно новые возможности. В частности, они позволяют ввести в расположение элементов третье измерение, перпендикулярное плоскости страницы. В этом и состоит смысл свойства **z-index:n**, где **n** — целое число. Чем больше значение **z-index**, тем выше в иерархии слоев (ближе к зрителю) располагается данный слой. Приведем пример (рис. 10), в котором те же изображения из файлов bild.jpg и class.gif частично наложены друг на друга за счет изменения значений соответствующих свойств **left, top, width, height:**

```
<div style="position:absolute; left:142px; top:82px;
width:152px; height:145px; z-index:2">
<img src="class.gif" width="154" height="148">
</div>
<div style="position:absolute; left:45px; top:37px;
width:167px; height:136px; z-index:1">
<img src="bild.jpg" width="180" height="135">
</div>
```

Изображение из файла class.gif, помещенное в первом из слоев, располагается над изображением (ближе к зрителю) из файла bild.gif потому, что первый слой имеет **z-index:2,** а следующий — **z-index:1.**

В следующем примере (рис. 11) продемонстрируем наложение слоя с текстом на слой, включающий изображение:

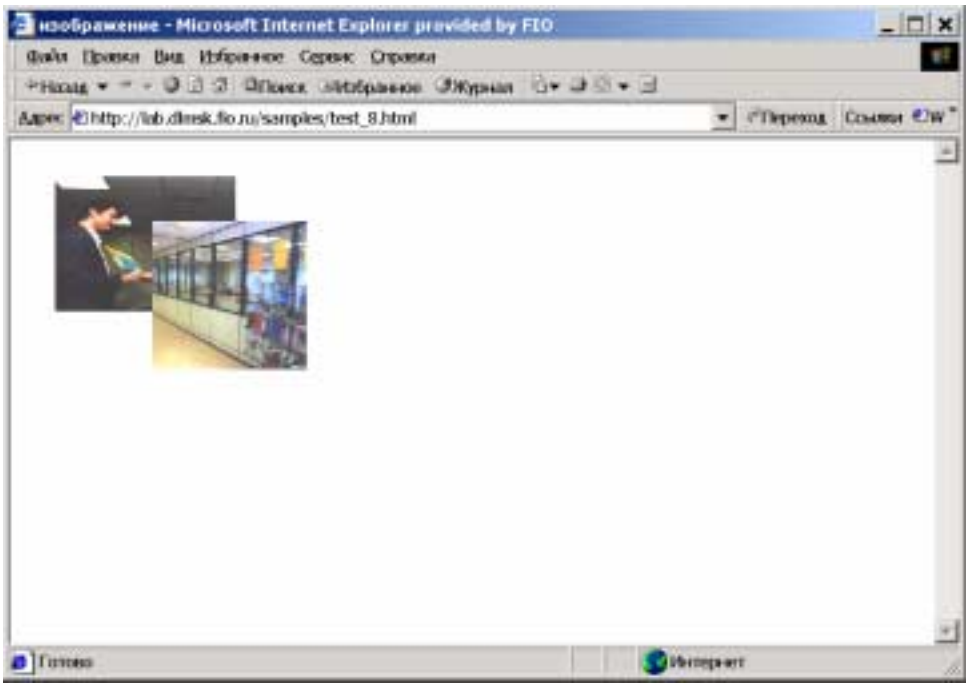

Рис. 10. Наложение слоев с разными значениями свойства z-index

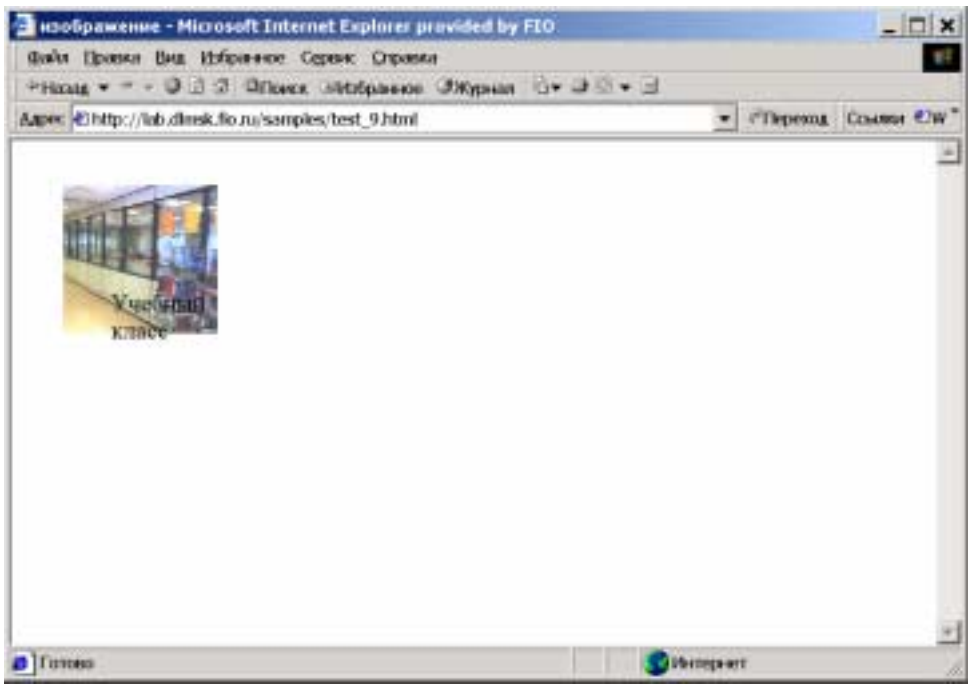

Рис. 11. Наложение слоя с текстом на слой с изображением

```
<div style="position:absolute; left:52px; top:45px;
width:154px; height:148px; z-index:1">
<img src="class.gif" width="154" height="148">
</div>
<div style="position:absolute; left:100px; top:150px;
width:121px; height:23px; z-index:2;
layer-background-color: #CCFFCC; border: 1px none #000000">
Учебный класс
</div>
```

В данном примере обратите внимание на второй слой. В нем задан фоновый цвет **layer-background-color: #CCFFCC** и располагается текст. Можно изменить порядок наложения слоев, изменив значение свойства **z-index**.

Мы здесь привели только начальные сведения о слоях. Для получения более подробной информации следует обратиться к специальной литературе. Отметим, что технология слоев по-разному разрабатывалась для браузеров Microsoft Internet Explorer и Netscape Navigator. Приведенные выше примеры выполнены в Internet Explorer. Со стандартными возможностями CSS можно познакомиться по адресу: **http://www.w3.org/TR/REC-CSS2**

Мета-теги

С HTML-документами работают в основном два типа программ: браузеры, показывающие содержание документа, и программы работающих в сети Интернет поисковых систем. Для программ поисковых систем HTML-документ должен содержать служебную информацию, которая вносится в документ с помощью мета-тегов и вводится с помощью атрибутов **alt.** Рассмотрим пример с мета-тегами.

```
<html>
<head>
<title>пример мета-тегов</title>
<meta http-equiv="Content-Type"
content="text/html; charset=windows-1251">
<meta http-equiv="Author" content="Федор Вирин.">
<meta http-equiv="KEYWORDS" content="маркетинг, раскрутка, вебмастер, сайт,
интернет, мастер, библиотека, новости, бесплатный, free, Webmaster, работа,
заработок, аналитика, аналитический отчет">
<meta http-equiv="DESCRIPTION" content="Библиотека мастера - это большая
коллекция материалов для вебмастеров по любой теме. Здесь можно найти любую
помощь для вашего сайта, вам помогут советом и делом." >
</head>
```

Мета-теги располагаются внутри элемента **head**, поэтому не видны пользователю при отображении документа браузером. В нашем примере четыре мета-тега. Каждый из них вводится служебным словом **meta**. Вслед за ним размещены атрибуты **http-equiv="имя"** и **content="содержимое".** Имя указывает, что именно описывает данный мета-тег. Атрибут **http-equiv="Author"** указывает, что мета-тег вводит имя автора документа. Атрибут **http-equiv="KEYWORDS"** указывает, что мета-тег содержит перечень ключевых слов документа. Мета-тег с атрибутом **http-equiv="DESCRIPTION"** содержит краткое описание документа. Из этих мета-тегов программа поисковой системы извлекает и сохраняет в своих базах данных соответствующую информацию. В дальнейшем пользователь, обратившись к поисковой системе и производя поиск, может найти документ или по имени автора, или по ключевым словам, а в результате поиска получить краткое описание документа.

Еще один очень важный мета-тег сообщает браузеру, в какой кодировке символов хранится документ:

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

Мета-тег указывает, что в данном случае текстовый документ хранится в кодировке **windows-1251**. Если бы текстовый документ был размечен средствами HTML и закодирован с использованием кода KOI8, то мета-тег имел бы вид

```
<meta http-equiv="Content-Type" content="text/html; charset=KOI8-R">
```

Существует большое количество разных вариантов мета-тегов, которые мы не описываем здесь. Приведем, к примеру, еще один полезный и интересный мета-тег:

```
<meta http-equiv ="Refresh" content="3, URL=http://www.name.ru/page.html">
```

Этот мета-тег определяет задержку в секундах (в данном случае — 3 секунды), после которой браузер автоматически вместо текущего документа загрузит страницу, находящуюся по адресу: **http://www.name.ru/page.html**

Дополнительную информацию о мета-тегах можно получить, например, в документе по адресу: **http://www. citforum.ru/internet/search/metatags.shtml**

Формы и CGI-протокол

Формы — важный элемент языка HTML, позволяющий посетителю Web-сайта передавать информацию Web-серверу. Таким образом, формы являются средством интерактивности. На рис. 12 пример отображения HTML-страницы с формой, которую предлагается заполнить пользователю.

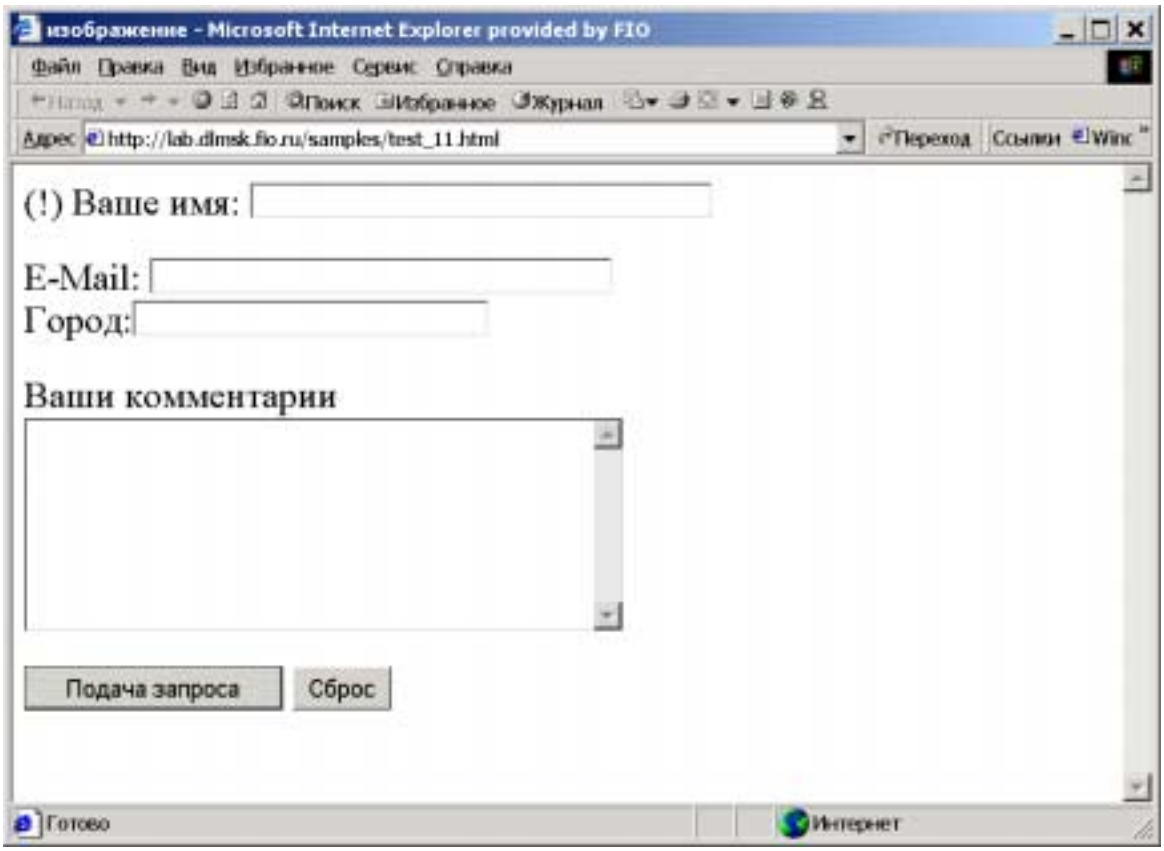

Рис. 12. Форма в рабочем поле браузера

HTML-текст, соответствующий форме на рис. 12:

```
<form method="POST" action="/cgi-bin/prog1.pl">
(!) Ваше имя: <input type="text"
name="realname" size="40"><p>
E-mail: <input type="text" name="email" size="40">
<br>Город:<input type="text" name="city" size="30">
<p> Ваши комментарии <br>
<textarea name="comments" cols="40" rows="8">
</textarea><p>
<input type="submit" name="submit" value= "Подача запроса">
<input type="reset" name="reset" value= "Сброс">
</form>
```

Разберем этот HTML-код. Форма задается элементом **form** с двумя атрибутами, которые указывают, каким образом (значение атрибута method) и какой обрабатывающей программе (значение атрибута **action**) отсылаются данные из формы. Внутри элемента **form** помещены обычный текст с элементами разметки **p** и **br** (например, **«Ваши комментарии»**), четыре элемента **input** и один элемент **textarea**. Элементы **input** с атрибутом **type="text"** формируют при отображении «поля ввода» в виде строк, длина которых (в условных единицах) определяется значениями соответствующих атрибутов **size**. Элемент **input** с атрибутом **type="submit"** создает изображение кнопки отправки информации от формы на экране браузера к Web-серверу. Атрибут **value="Подача запроса"** определяет надпись на изображении кнопки. Элемент **input** с атрибутом **type="reset"** отображается в виде кнопки сброса (очистки) полей формы (например, при неверном заполнении поля). Атрибут **value="Сброс"** определяет надпись на изображении кнопки сброса. Элемент **textarea** формирует при отображении «многостроковое» поле ввода. Атрибут **rows** указывает количество строк, атрибут **cols** определяет длину каждой строки (в символах). Атрибуты **name** во всех перечисленных элементах служат для указания имен, которые используются в данных, пересылаемых от формы к Web-серверу. Именно по этим именам программа prog1.pl определяет, в какое поле ввода какую информацию ввел пользователь. В данном примере текст, внесенный в поля **«Ваше имя», «E-mail», «Город»** и **«Ваши комментарии»**, при нажатии кнопки **«Подача запроса»** будет отправлен программе cgi-bin/prog1.pl методом **post** на сервер. К сожалению, понятное объяснение особенностей метода **post** требует знакомства с протоколом HTTP (Hypertext Transfer Protocol) — сетевым протоколом, специально созданным для World Wide Web, что выходит за рамки данного обзорного пособия. Программа prog1.pl, находящаяся на Web-сервере в каталоге /cgi-bin, анализирует полученные данные (рис. 13)

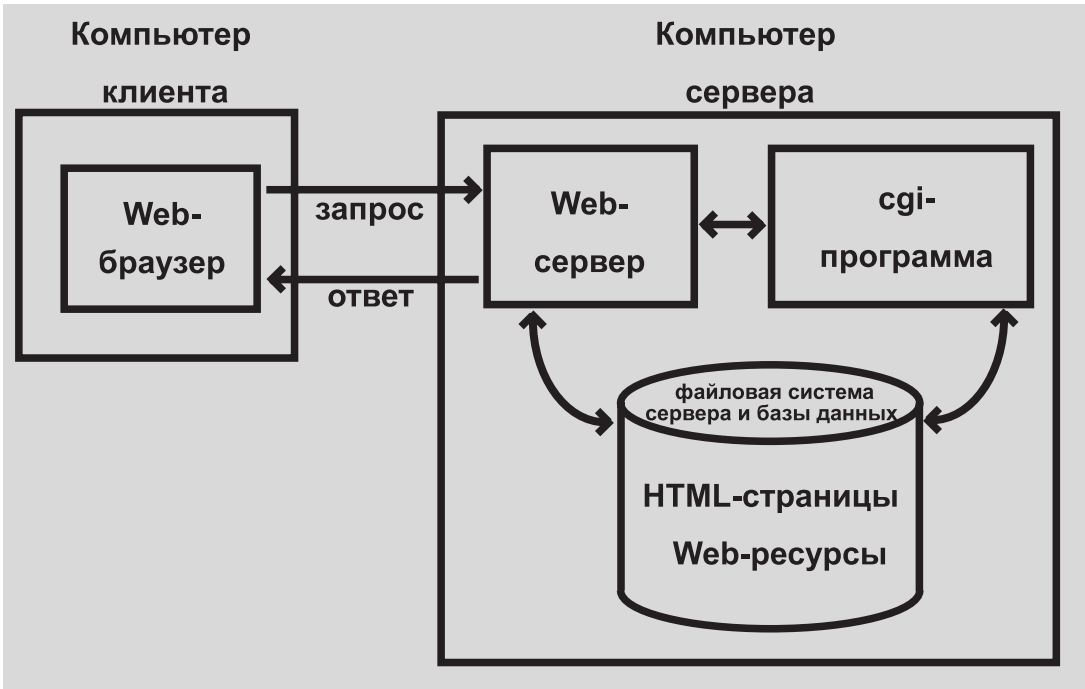

Рис. 13. Взаимодействие браузера, сервера и CGI-программы

и либо записывает их в какую-либо базу данных, либо формирует и отправляет ответ пользователю. Такой механизм обмена данными между клиентом и программами, выполняемыми на сервере, регламентируется протоколом, который называется «унифицированный шлюзовой интерфейс» — Common Gateway Interface, сокращенно CGI. Обычно CGI применяется в Web-серверах, функционирующих на платформе UNIX, и в классических Web-серверах (например, Apache), которые перенесены и в другие операционные системы (Windows, MacOS). На базе OS Windows с использованием Web-сервера IIS (Internet Information Server) используется и другой механизм — «активные серверные страницы» (ASP — Active Server Pages). Аналогом механизма ASP служит технология PHP, которая применима не только на серверах фирмы Microsoft.

Важно понимать, что наличие на Web-странице формы само по себе не обеспечивает интерактивности. Необходимо наличие на сервере программы, в нашем примере она имеет название prog1.pl, обрабатывающей данные, передаваемые браузером от формы (см. рис. 13).

Мы рассмотрели здесь только один пример форм, чтобы продемонстрировать общие принципы их создания. Существует большое многообразие различных HTML-средств для создания разных форм, но их изучение требует знакомства не только с Web-протоколами, но и с языками программирования, используемыми для написания CGI-программ, выполняемых на серверной стороне. Важно отметить, что при разработке и отладке форм и CGI-программ Web-сервер и Web-браузер могут размещаться на одной и той же ЭВМ, что отличается от схемы, изображенной на рис. 13. В этом случае можно считать, что рис. 13 содержит схему логического взаимодействия клиента и сервера, которые физически исполняются на одной ЭВМ.

Технология Server Side Include (SSI)

Server Side Include в переводе означает «включаемый на стороне сервера». Эта технология позволяет Web-серверу включать в текст HTML-страниц любой другой текст: содержимое текстовых файлов или, например, результат работы программ на сервере. Происходит это «включение» непосредственно перед передачей текста HTML-страницы браузеру пользователя. Чтобы ясно понимать действие и назначение этого механизма, нужно помнить, что HTML-страницы размещаются в файловой системе серверного компьютера (см. рис. 13). Доступ к HTML-страницам на серверной стороне Web-браузер имеет только опосредованный. Запрос Web-браузера отсылается Web-серверу, а Web-сервер по указанному адресу (URL) «выбирает» нужную страницу и отсылает ее Web-браузеру для отображения. Перед пересылкой HTML-страницы Web-сервер «прочитывает» ее текст и, обнаружив директивы SSI, выполняет их. Что дает технология SSI?

Предположим, на каждой странице сайта вы поместили список его разделов. Сайт, как и положено, постоянно развивается, и, допустим, надо добавить еще один раздел. Если не прибегать к специальным средствам, то придется исправлять каждую страницу сайта, содержащую этот список. Если же использовать SSI, достаточно создать отдельный текстовый файл (например, с именем menu.html) и занести туда HTML-код списка разделов. Непосредственно в страницы сайта, в те места, где должен находиться список разделов, нужно вставить следующую инструкцию (SSI-директиву):

```
<!--#include virtual="menu.html"-->
```

При запросе этой страницы сервер заменит указанную инструкцию **#include** содержимым файла menu.html. Если изменить файл menu.html, то все страницы, где использовалась инструкция **#include**, обновятся автоматически. Главное в том, что в HTML-файлах, лежащих на диске Web-сервера, ничего не меняется — сервер производит вставку текста «на лету», перед тем, как отправить HTML-страницу браузеру клиента. Вместо имени обычного текстового файла в инструкции **#include** можно указывать имя программы, результаты выполнения которой нужно вставить в текст HTML-страницы. Так можно реализовать, например, счетчик посещений, оповещающий клиента о том, каким по счету посетителем сайта он является.

Необходимо понимать, что невозможно экспериментировать с механизмом SSI на машине, на которой не установлен Web-сервер. Функционирование SSI обеспечивается именно сервером, поэтому при просмотре HTML-файлов с локального жесткого диска браузер проигнорирует директивы SSI — ведь они, с точки зрения HTML, являются обычными комментариями (заключены в скобки <!- -...- ->). Кроме того, сервер должен быть настроен на обработку SSI-директив.

При включении указанной настройки на исполнение SSI-директив сервер просматривает все страницы, разыскивая в них SSI-директивы, что увеличивает нагрузку на сервер. Чтобы не выполнять эту работу для обычных HTML-страниц, для страниц имен файлов с использованием SSI дается специальное расширение — обычно .shtml, при наличии которого Web-сервер автоматически выполняет обработку SSI-директив в тексте из этого файла.

Отметим, что директива **#include** не единственная SSI-директива и возможности технологии SSI достаточно широки.

Скрипты в HTML-документах

Язык HTML предоставляет авторам Web-страниц широкие возможности для отображения текстовой и графической информации и включения в состав страниц различных объектов. Но, тем не менее, создаваемые с помощью языка HTML-страницы остаются статическими — пользователи, не выходя за пределы HTML-средств, не могут изменять информацию, расположенную на странице, и даже использовать большинство интерфейсных элементов. Для того, чтобы сделать страницу по-настоящему интерактивной, нам нужен язык, выполняемый в контексте браузера. Такой язык называют скриптовым.

Скриптовый язык обычно не содержит всех возможностей настоящих языков программирования. Созданные с помощью скриптовых языков программы (называемые скриптами или сценариями) после включения в HTML-страницу не могут выполняться самостоятельно — они работают только в контексте браузера, поддерживающего их выполнение. Скрипты включаются в состав Web-страниц и распознаются и обрабатываются браузером при отображении остального HTML-кода той же страницы.

Web-страница, содержащая скрипт или несколько скриптов, может обрабатывать события, связанные с окном браузера, — такие, как загрузка документа, закрытие окна, появление курсора над некоторым объектом страницы, нажатие кнопки «мышь» или клавиши клавиатуры и т. п. При этом в виде скриптов могут быть реализованы обработчики разных событий.

В настоящее время существуют два языка для написания скриптов для Web-страниц — JavaScript и Visual Basic Script. Visual Basic Script — это разработка Microsoft. Скрипты на этом языке выполняются пока только в браузере Internet Explorer (VBScript). JavaScript был первоначально разработан компанией Netscape, а затем Microsoft разработала свою версию этого же языка, названную JScript. К сожалению, это привело к возникновению разных, не всегда совместимых, версий языка JavaScript.

Код JavaScript начинается обычно тегом **<script language="JavaScript">**. С его помощью браузеру указывается, что далее следует скрипт на языке JavaScript. Далее располагается сама скриптовая программа. Закрывающий тег строится по обычным правилам: **</script>**.

Пример HTML-документа со скриптом на языке JavaScript, вычисляющим сумму двух чисел:

```
<html>
<head>
<script language="JavaScript">
function calculation()
{
var x= 12;
var y= 5;
var result= x + y;
alert ("Result="+);
}
</script>
</head>
<body>
```

```
<form>
<input type="button" value="Calculate"
onClick="calculation()">
</form>
</body>
</html>
```

Как и при обработке обычной формы, на экране изображается кнопка с названием «Calculate». При нажатии на нее («мышкой») выполняется JavaScript — функция calculation(), из которой вызывается alert(). Это приводит к формированию в окне браузера модальной кнопки с результатом (рис. 14).

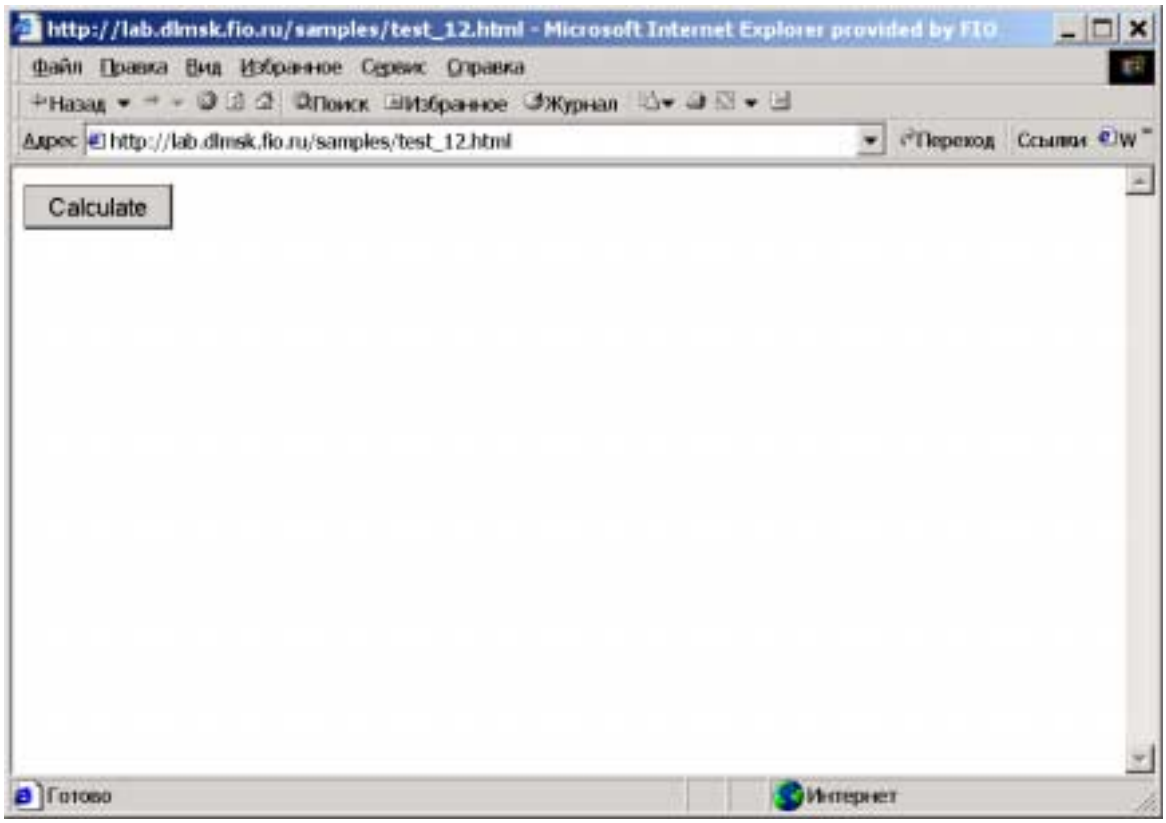

Рис. 14. Отображение HTML-документа с JavaScript-кодом

Динамический HTML (DHTML)

Несмотря на то, что сценарии (скрипты), исполняемые на клиентской стороне, увеличивают гибкость HTML, они мало используются в целях более серьезных, нежели проверка данных в форме перед ее «отправкой» Web-серверу. Ограниченность сценариев заложена в самой природе HTML. HTML-текст поступает в программу просмотра посимвольно, и эта программа (браузер) «конструирует» и отображает Web-страницу по мере этого поступления. Как только символы текста HTML-документа прекращают поступать к программе просмотра, «конструирование» заканчивается и HTML-страницу уже нельзя изменить.

Программа просмотра (браузер), исполняя программу сценария (скрипта), может только в ограниченной мере модифицировать HTML-текст по мере того, как он поступает к ней, либо может целиком заменить отображенную страницу новой.

Для преодоления этого ограничения Microsoft предложила технологию DHTML. Dynamic HTML позволяет изменять содержимое HTML-страницы при помощи скриптов *после* окончания конструирования. Такая возможность появляется за счет того, что каждый элемент HTML-документа рассматривается как объект со своими свойствами. Эти свойства доступны в коде скрипта на JavaScript или VBScript. Интерактивно взаимодействуя с отображением документа, пользователь посредством скриптов изменяет свойства объектов, например, значения атрибутов элемента или свойства, введенные листами стилей. Браузер на основе новой информации изменяет отображение документа.

Dynamic HTML строится на основе JavaScript или VBScript, причем функционирование скриптов (сценариев) соответствует парадигме использования программ, управляемых событиями. Это означает, что код скрипта выполняется с учетом взаимодействия пользователя с элементами Web-страницы. Основная особенность этой схемы состоит в том, что поддерживать взаимодействие с пользователем в Dynamic HTML может любой элемент Web-страницы. Таким образом, DHTML обеспечивает автора Web-документов средствами программирования, позволяющими полностью управлять HTML-документом на стороне клиента.

Технология активных серверных страниц (ASP)

Dynamic HTML представляет собой средство программирования для Microsoft Internet Explorer. Однако такие браузеры, как, например, Netscape Navigator, поддерживают другой диалект Dynamic HTML. «Заставить» браузеры разных фирм работать по одним правилам на сегодняшний день практически невозможно. Это служит препятствием для создания кросс-платформенных Web-страниц.

Чтобы разработать Web-сайт, доступный самым различным программам просмотра, необходимо перенести программирование с клиента на сервер. Такую возможность предоставляют несколько конкурирующих технологий. Рассмотрим вначале одну из них — Microsoft ASP (Active Server Pages — активные серверные страницы). По сути, технология ASP — не что иное, как встраивание в текст Web-документа сценария (скрипта), который исполняется на сервере. Когда клиент запрашивает Web-страницу, этот сценарий порождает HTML-текст, который воспринимает любая программа просмотра. Сценарий в ASP-технологии программируется обычно на VBScript или JavaScript (JScript). Ключевая особенность ASP-сценария из Web-страницы состоит в том, что он исполняется на серверном компьютере, а клиенту отсылается только HTML-текст (без кодов на языке скриптов).

Следующий пример демонстрирует простую ASP-страницу со скриптом на языке VBScript, которая формирует HTML-страницу с приветствием в соответствии со временем суток. В этом примере текст приветствия формируется как значение строковой переменной strGreeting. Текущий час определяется при помощи выражения Hour(Now), где Now — функция VBScript, возвращающая текущий момент времени и дату. Если значение часа меньше 12, то приветствие задается в форме «Good Morning!» («Доброе утро!»). От полудня до шести вечера сообщение имеет вид «Good Afternoon!» («Добрый день!»), а после шести — «Good Evening!» («Добрый вечер!»).

```
<%@ LANGUAGE="VBSCRIPT" %>
<html>
<head>
<title>Simple ASP Example</title>
</head>
<body>
<%
Dim strGreeting
If Hour(Now) < 12 Then
    strGreeting = "Good Morning!"
ElseIf Hour(Now) > 11 And Hour(Now) < 18 Then
    strGreeting = "Good Afternoon!"
ElseIf Hour(Now) > 17 Then
    strGreeting = "Good Evening!"
End If
%>
<h1><%=strGreeting%></h1>
</body>
</html>
```

После того, как переменная strGreeting получила значение, текст помещается в элемент **h1** с помощью выражения **<%=strGreeting%>**. На этом интерпретация скрипта заканчивается и HTML-текст сформирован. Таким образом, если системное время на сервере, например, больше 17 часов, то браузер посетителя получит следующий HTML-документ:

```
<html>
<head>
<title>Simple ASP Example</title>
</head>
<body>
<h1> Good Evening!</h1>
</body>
</html>
```

Резюмируем сказанное о технологии ASP. Active Server Pages позволяет встраивать в Web-документ скрипт (сценарий) и выполнять его на сервере. Концепция очень проста. Сценарий, который нужно выполнить на сервере, размещается внутри специальных скобок <%...%>. Они показывают, что находящийся внутри код необходимо выполнить на сервере и результат послать клиенту. Данный код сценария не доступен и не виден пользователю. Сам код сценария может быть написан с использованием VBScript, а также JavaScript (или JScript). Что происходит при обращении к ASP-странице?

Пользователь вводит адрес ASP-страницы (имя файла с расширением **.asp**) в адресную строку браузера и нажимает Enter для запроса данной страницы. Далее выполняются следующие шаги:

- браузер посылает Web-серверу запрос на получение ASP-файла (ASP-страницы);
- Web-сервер получает запрос и по расширению **.asp** в имени файла распознает, что запрашивается ASP-страница;
- Web-сервер считывает нужный файл с диска;
- Web-сервер передает данный файл специальной программе-интерпретатору ASP-файлов;
- текст ASP-файла обрабатывается «сверху вниз». Все встречающиеся команды скриптового языка выполняются. В результате обработки получается стандартный HTML-файл.

Этот HTML-файл отсылается браузеру.

Технология ASP в MS Windows поддерживается Web-серверами IIS (Internet Information Server) и PWS (Personal Web Server). Существуют продукты сторонних компаний, поддерживающие технологию ASP на других платформах. Например, Chili!Soft, Inc. Chili!Soft ASP (**http://www.chilisoft.com/**). Halcyon Software, Inc. разработала «Instant ASP» (**http://www.halcyonsoft.com/**).

Технология PHP

Чтобы у читателя не сложилось впечатления, что почти все средства, применяемые в качестве элементов Web-страниц, или созданы, или внедрены, или освоены, или распространяются силами фирмы Microsoft, кратко расскажем о технологии PHP.

Названная технология включает в себя язык программирования (PHP), интерпретатор этого языка, средства реализации CGI-протокола и библиотеку функций, обеспечивающих доступ к разным ресурсам Интернет. В целом PHP имеет практически те же возможности, что и технология ASP, разработанная фирмой Microsoft. Однако PHP работает не только с узким кругом Web-серверов Microsoft, и в этом несомненное достоинство PHP.

Как утверждает «Руководство по PHP 3.0» (на русском языке доступно по адресу: **http://www.webclub.ru/content/programming_php/article-69.html)**, самая значимая возможность PHP3 — средства интеграции с базами данных. В настоящее время (май 2001 г.) современной является версия PHP 4.0.4. С ее преимуществами можно познакомиться, обратившись по адресу: **http://www.php.net**, но в данном ознакомительном пособии достаточно рассмотреть самые общие возможности PHP.

Что касается собственно языка PHP, то это кросс-платформенный интерпретируемый на стороне Web-сервера язык программирования, предназначенный для создания активных Web-страниц. Синтаксис языка PHP основан на принципах построения языков C, Perl, Java. Код скрипта (сценария) на языке PHP встраивается непосредственно в HTML-текст. При запросе клиентом HTML-страницы со скриптом на языке PHP Web-сервер выполняет интерпретацию операторов языка. Обычно указанная интерпретация предусматривает изменение исходного HTML-текста, и этот измененный текст отправляется клиенту.

Чтобы «заставить» Web-сервер обращать внимание на PHP-код, включенный в HTML-страницу, чаще всего бывает достаточно записать эту страницу в текстовый файл с названием, имеющим расширение, например «**.php**». Сам код PHP-скрипта или отдельного PHP-оператора выделяется в тексте HTML-документа специальными скобками <?php ... ?>. Текст вне этих скобок PHP-интерпретатор не рассматривает, и он передается браузеру (клиенту) без изменений.

Пример HTML-документа, включающего PHP-код (на основе теста для Web-сервера OmniHTTPd/2.08):

```
<!-- MyTest.php — пример на основе теста из OmniHTTPd/2.08 -->
<html>
    <head>
    <title>Example PHP Script</title>
    </head>
<body>
<h3>Simple Echo</h3>
<?php echo "Hi, I'm a PHP script!"; ?>
<h3>Server Variables</h3>
<b>SCRIPT_NAME:</b> <?php echo $SCRIPT_NAME; ?><br>
```

```
<b>QUERY_STRING:</b>
<?php echo $QUERY_STRING; ?><br>
<b>SERVER_SOFTWARE:</b>
<?php echo $SERVER_SOFTWARE; ?><br>
<p>
For more information on programming with PHP,
<br>check out the official PHP page at
<a href="http://www.php.net">www.php.net</a>.
</p>
</body>
</html>
```

После обработки приведенного текста Web-сервером браузер получит обыкновенный HTML-документ, не содержащий PHP-кодов. Вот текст этого HTML-документа, который можно увидеть на стороне клиента:

```
<!-- MyTest.hpp — пример на основе теста из
OmniHTTPd/2.08 -->
<html>
    <head>
    <title>Example PHP Script</title>
    </head>
<body>
<h3>Simple Echo</h3>
Hi, I'm a PHP script!
<h3>Server Variables</h3>
<b>SCRIPT_NAME:</b> /MyTest.php<br>
<b>QUERY_STRING:</b> <br>
<b>SERVER_SOFTWARE:</b>
OmniHTTPd/2.08<br>
<p>
For more information on programming with PHP,
<br>check out the official PHP page at
<a href="http://www.php.net">www.php.net</a>.
</p>
</body>
</html>
```

Результат (рис. 15) отображения браузером такого HTML-текста для читателя должен быть совершенно понятен, если он знаком с именами и назначением переменных среды окружения SCRIPT_NAME, QUERY_STRING, SERVER_SOFTWEARE. Для целей настоящего ознакомительного пособия достаточно указать, что имена переменных среды окружения фиксированы, их значения — символьные строки.

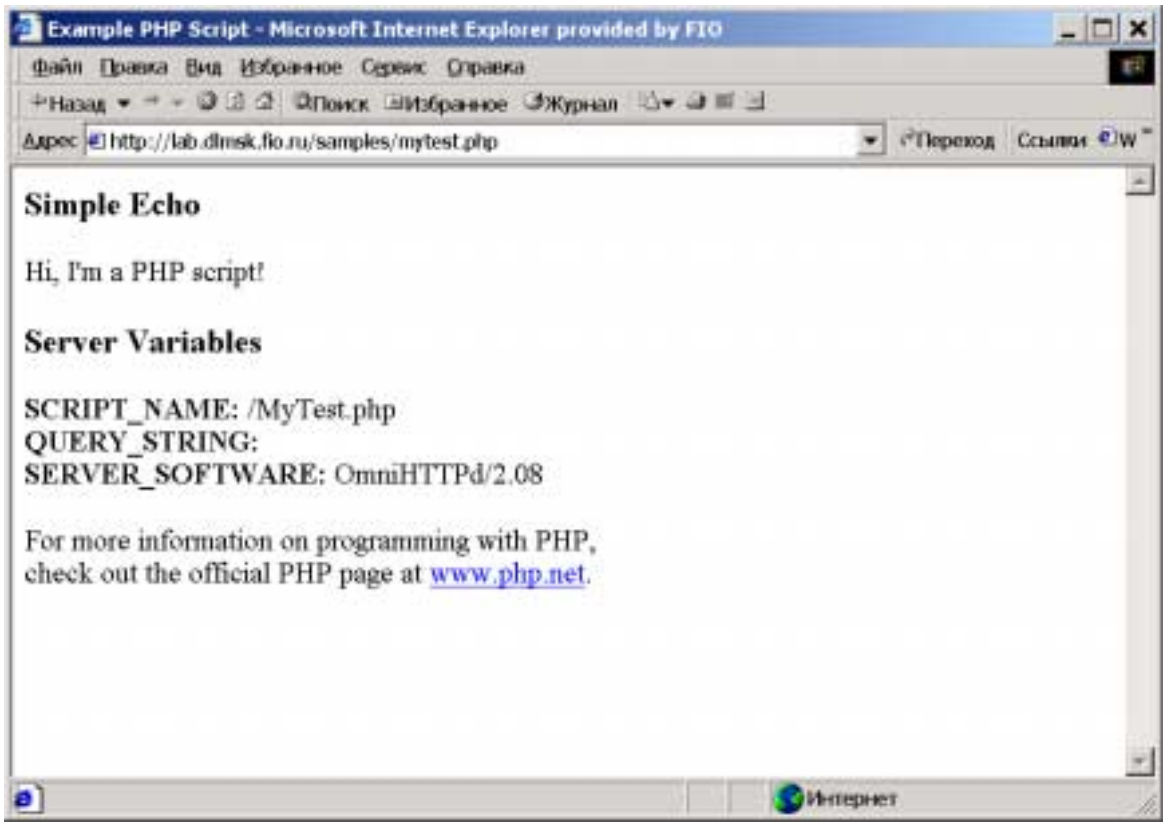

Рис. 15. Отображение HTML-страницы, сформированной с применением PHP-операторов

При выполнении PHP-кода на серверной стороне значения переменных среды окружения доступны с помощью обозначений:

$SCRIPT_NAME — имя интерпретируемого PHP-документа;
$QUERY_STRING — переданная от браузера информация;
$SERVER_SOFTWARE — имя Web-сервера, под управлением которого интерпретируется PHP-скрипт.

При интерпретации PHP-документа каждый оператор вида <?php echo строка;?> включает в формируемый HTML-текст значение строки. Если строка задана в виде константы, например <?php echo "Hi, I'm a PHP script!"?>, то она выводится непосредственно. Вместо имен переменных среды окружения включаются их значения. Например, вместо $SCRIPT_NAME будет использована строка «MyTest.php». Остальное очевидно.

Java-аплеты

В 1995 г. фирмой SUN Microsystems был разработан язык программирования Java. Одной из главных особенностей этого языка является его независимость от платформы (от операционной системы и аппаратных средств ЭВМ), на которой выполняются программы, написанные на Java. Это обстоятельство сделало применение Java-программ в Интернете весьма привлекательным и перспективным.

Язык Java является объектно-ориентированным, многопотоковым, динамическим и так далее, но вовсе не эти свойства превращают его в мощный язык для сетевого программирования. Главное в том, что Java-программы исполняются в *виртуальной машине*, если таковая имеется в операционной системе компьютера. (Виртуальная машина — это программный модуль, интерпретирующий код Java-программы.) Java-программа как таковая не имеет никакого контакта с настоящим, физическим компьютером, все, с чем она взаимодействует, — это виртуальная машина. Такой подход приводит к некоторым важным особенностям.

Во-первых, как уже отмечалось выше, Java-программы настолько мобильны, что не зависят от платформы, на которой они исполняются. Java-программа без каких-либо изменений будет работать на любой ЭВМ, где есть виртуальная машина. Другими словами, Java-программа всегда пишется только для единственной платформы — для виртуальной машины. Говоря о мобильности, можно сравнить язык Java с другими языками программирования.

Язык ANSI C, например, тоже не зависит от платформы, однако программы на нем не являются в полном виде переносимыми — их необходимо каждый раз компилировать заново на каждой новой платформе. Кроме того, язык ANSI C оставляет размеры и форматы внутренних структур данных на усмотрение разработчиков конкретной операционной среды — в Java же все они заранее строго определены и неизменны.

Программы на языках типа C или C++ запускаются на исполнение напрямую операционной системой. Поэтому они получают прямой доступ к системным ресурсам компьютера, включая оперативную память и файловую систему.

Поскольку Java-программы запускаются виртуальной машиной, ее разработчики и решают, что разрешено, а что запрещено делать программе. Окружение, в котором работает Java-программа, называется оболочкой времени выполнения (runtime environment). Виртуальная машина играет роль бастиона на пути между Java-программой и информационными ресурсами компьютера, на котором та выполняется. Java-программа никогда не сможет получить прямой доступ к устройствам ввода-вывода, файловой системе и даже памяти. Вместо Java-программы все это делает виртуальная машина.

Java-программы разделяются на два класса: программы-приложения, т. е. полнофункциональные программы, которые могут обращаться (через виртуальную машину) к файловой системе компьютера, устанавливать сетевые соединения и т. п.; Java-аплеты, предназначенные для встраивания в Web-страницы. Естественно, на них накладываются серьезные ограничения, связанные с безопасностью для клиента.

В Web-системе наибольшее распространение получили аплеты, т. е. Java-программы, исполняемые только на стороне Web-клиента в рамках работы браузера (рис. 16). Когда загружается и запускается аплет, виртуальная машина полностью запрещает ему доступ к файловой системе. Виртуальная машина разрешает Java-аплету только косвенный

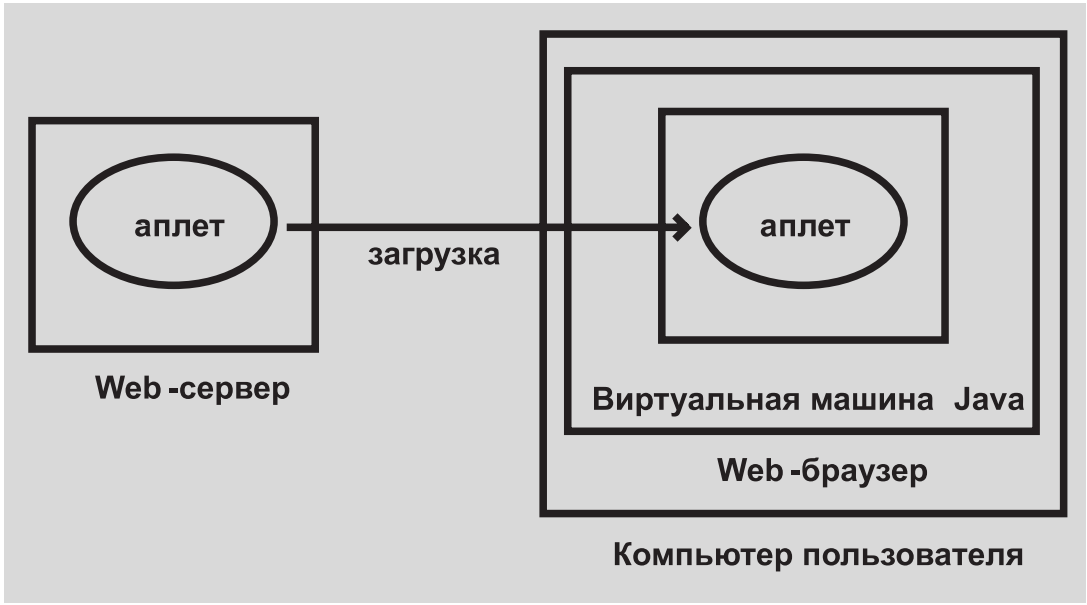

Рис. 16. Java-аплет в Web-системе

доступ к избранным системным ресурсам. Вот почему аплетам, которые клиент получит из сети Интернет, можно доверять: они не способны уничтожать файлы или распространять вирусы.

Виртуальная Java-машина и вся Java-технология позволяют Java-программе собираться «по кусочкам» прямо в процессе выполнения (интерпретации). Это практично, поскольку наиболее важные части программы можно постоянно хранить в памяти, а менее важные загружать по мере необходимости. Виртуальные Java-машины «умеют» делать это, пользуясь механизмом «динамического связывания» (dynamic binding).

Сами аплеты, включенные в HTML-страницы, — это файлы, содержащие так называемый байт-код (который исполняется виртуальной Java-машиной). Имена файлов с байт-кодами аплетов имеют расширение **.class**.

Для встраивания вызовов аплетов в текст HTML-документа в язык HTML был введен элемент **applet**, который начинается тегом **<applet>** и кончается тегом **</applet>**. Элемент **applet** решает две задачи. Во-первых, он обеспечивает вызов Java-аплета из HTML-документа. Во-вторых, при отображении документа элемент **applet** отводит место для размещения формируемой аплетом информации в рабочем поле браузера. В общем виде документ, содержащий ссылки на аплеты, будет выглядеть так, как это представлено в следующем примере:

```
<html>
<head>
<title>Документ со встроенной ссылкой на aplet.</title>
```

```
</head>
<body>
<applet code= "hello.class" width=200 height=100>
</applet>
</body>
</html>
```

В начальном теге элемента **applet** указывается поле шириной 200 пикселов и высотой 100 пикселов, выделяемое для отображения результатов выполнения аплета. Элемент **applet** загружает аплет с именем hello.class. Файл hello.class должен в этом случае находиться в том же каталоге, что и HTML-файл, в котором есть на него ссылка. После получения байт-кода аплета браузер отведет место для отображения результатов аплета в своей рабочей области и только после этого начнет его исполнение. (Исполняет байт-код аплета виртуальная Java-машина, включенная в браузер.)

В общем случае элемент **applet** имеет следующий вид:

```
<applet [codebase = codebase_url]
code = "aplet.class"
[alt = text]
[name= aplet_name]
width = number_of_pixels
height = number_of_pixels
[align = alignment]
[vspace=number_of_pixels]
[hspace=number_of_pixels]>
[<param name=param_name value=param_value>]
[HTML text]
</applet>
```

Атрибут **codebase** задает базовую часть адреса для файла с байт-кодом аплета. Атрибут **code** вводит имя этого файла аплета, которое должно иметь расширение .class. Атрибут **alt** вводит альтернативный текст, который отображается браузером в том случае, когда выполнение аплета на машине клиента запрещено. Атрибут **name** — имя элемента **applet**, используется для ссылки на него. Атрибут **width** — ширина области, выделяемой в рабочем поле браузера для отображения аплета. Атрибут **height** — высота этой же области отображения аплета. Атрибут **align** управляет горизонтальным выравниванием области отображения аплета внутри рабочей области браузера. Атрибуты **vspace** и **hspace** задают величины отступов от области отображения аплета до текста HTML-документа (вертикальный и горизонтальный соответственно). Из атрибутов элемента **applet** обязательными являются только **code, width** и **height**. Все остальные атрибуты (они заключены в квадратные скобки [ ]) можно опускать.

В элемент **applet** как в контейнер могут вкладываться элементы **param** и HTML-текст. Использование элементов **param** позволяет передавать параметры внутрь аплета из HTML-документа. Этот механизм аналогичен использованию команды с несколькими аргументами.

Встраивание объектов. Элемент OBJECT

Элемент **object** позволяет вставлять в Web-страницу такие объекты, которые браузер самостоятельно не распознает. Рассмотрим пример, текст которого приведен ниже. Пусть в Web-страницу нужно вставить объект, созданный во флэш-технологии, например, анимированное изображение во флэш-формате. Предположим, что код изображения находится в файле Movie1.swf. Браузер не может работать с данными этого формата, однако к браузеру можно подключить вспомогательную программу для «проигрывания» флэш-файлов — флэш-плейер. Если такой плейер уже установлен, браузер распознает его наличие по идентификатору, указанному в качестве значения атрибута classid: **classid= "clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"**. Если плейер не установлен, то он автоматически устанавливается с сайта фирмы Macromedia. Для этого атрибут codebase в качестве значения определяет адрес (URL) плейера: **codebase= "http://active.macromedia.com/flash2/cabs/swflash.cab#version=4,0,0,0".** Далее указываются размеры анимированного изображения во флэш-формате на Web-странице: **width=250 height=200**. Затем идет перечисление параметров, которые передаются флэш-плейеру. Первый параметр **<param name=movie value="Movie1.swf">** указывает имя файла с кодом анимированного изображения: Movie1.swf. Следующий параметр задает качество воспроизведения: **<param name=quality value=high>**. Параметр **<param name=bgcolor value=#FFFFFF>** указывает фоновый цвет области анимации. Этого было бы достаточно, если бы в сети Интернет не присутствовали пользователи со старыми версиями браузеров. Некоторые версии браузеров «не умеют» обрабатывать тег **<object>**. В более ранней спецификации языка HTML присутствовал только тег **<embed>**, поэтому для этих браузеров рассматриваемый ниже пример следовало бы дополнить элементом **embed**. У этого элемента набор атрибутов отличен от атрибута элемента **object**, но они полностью дублируют информацию, заданную для загрузки объекта Movie1.swf в элементе **object**:

```
<html>
<head>
<title>Movie1</title>
</head>
<body>
<object
classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#version=4,0,0,0"
width=250 height=200>
<param name=movie value="Movie1.swf">
<param name=quality value=high>
<param name=bgcolor value=#FFFFFF>
</object>
</body>
</html>
```

Поясним назначение основных атрибутов элемента **object**: classid = url.

Этот атрибут может использоваться для указания местоположения объекта с помощью URL, указанного как значение атрибута: codebase = url.

Этот атрибут определяет базовую часть всех используемых в элементе относительных адресов URL. Если этот атрибут отсутствует, значением базовой части адреса объекта по умолчанию является базовый адрес текущего документа.

Переход к языку XML

(При написании настоящего раздела использованы материалы из статьи А. Печерского «Язык XML — практическое введение» — **www.citforum.ru/internet/xml/index.shtml**)

SGML (Standart Generalised Markup Language — стандартный обобщенный язык разметки) был утвержден ISO в качестве стандарта еще в 80-х годах (стандарт ISO 8879:1986). Этот язык, во-первых, служит универсальным средством разметки текстовой информации, а во-вторых, используется в качестве метаязыка для определения других языков разметки (текстов). Он определяет для каждого вводимого им языка допустимый набор тегов, их атрибуты и внутреннюю структуру документа. Контроль за правильностью использования элементов разметки в тексте, размеченном с помощью SGML, осуществляется при помощи специального набора правил, называемых DTD-описаниями (DTD — Document Type Definition — определение типа документа), которые используются программой обработки размеченного текста при разборе (например, при отображении) документа. Для каждого класса документов определяется свой набор правил, описывающих грамматику соответствующего языка разметки. С помощью SGML можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, и представлять эту информацию в некотором стандартизованном формате. Но ввиду своей сложности (свыше 500 страниц труднопонимаемых спецификаций) SGML используется, в основном, для описания синтаксиса других языков (наиболее известным из которых является HTML), и лишь немногие приложения работают с SGML-документами напрямую.

Гораздо более простой и удобный, чем SGML, язык HTML позволяет определять оформление элементов документа только с помощью ограниченного набора инструкций разметки — тегов, при помощи которых осуществляется процесс разметки. Инструкции HTML, в первую очередь, предназначены для управления процессом отображения содержимого документа на экране программы-клиента. Тем самым они определяют способ представления документа, но не его структуру. В качестве основы гипертекстовой информационной системы, описываемой HTML, используется текстовый файл, который может легко храниться в файловой системе и передаваться по сети. Эта особенность, а также тот факт, что HTML является открытым стандартом, в силу чего множество пользователей могут применять его для оформления своих документов, безусловно, повлияли на рост популярности HTML и сделали его на сегодня главным средством представления информации в Web.

Современные приложения, однако, нуждаются не только в языке представления данных на экране клиента, но и в механизме, позволяющем определять структуру документа и описывать соотношение содержащихся в нем элементов. HTML вполне успешно справляется с задачей описания текстовой информации и отображением ее на экране программы просмотра — браузера. Но структура отображаемых данных никак не связана с теми тегами, которые используются для форматирования, поэтому у программ-анализаторов нет возможности использовать теги HTML для поиска нужных фрагментов документа.

Таким образом, встретив, например, такое описание: **rose**, программа просмотра (браузер) может определить, каким цветом отобразить текст, содержащийся внутри тегов ****, и, вероятно, отобразит его правильно. Однако для браузера, интерпретирующего HTML-теги, не имеет значения, в каком месте документа встретился этот элемент, в какие другие теги заключен текущий фрагмент, существуют ли вложенные в него фрагменты и правильно ли построены отношения между

фрагментами документа. Такое «безразличие» к структуре документа приводит к тому, что поиск или анализ информации внутри него ничем не отличается от работы со сплошным, не разбитым на элементы текстовым файлом. А это, как известно, не самый эффективный способ работы с информацией.

Другим существенным недостатком HTML можно назвать ограниченность (фиксированность) набора его тегов. Их количество и состав закреплены спецификацией языка, и поэтому у разработчика нет возможности вводить собственные, специальные теги. Хотя время от времени появляются новые расширения языка (на сегодняшний день последней версией HTML является HTML 4.0), но долгий путь стандартизации, сопровождаемый постоянными разногласиями между основными производителями браузеров, делает практически невозможной быструю адаптацию языка HTML и его использование для отображения специализированной информации (например, мультимедийной или математических, химических формул и т. д.).

Подводя итог сказанному, можно утверждать, что сегодня HTML уже не удовлетворяет в полной мере требованиям, предъявляемым современными разработчиками к языкам подобного рода. И ему на смену был предложен новый язык разметки гипертекста — мощный, гибкий и одновременно с этим удобный язык XML. В чем же заключаются его достоинства?

XML (*eXtensible Markup Language — расширяемый язык разметки*) — это язык разметки, описывающий целый класс информационных объектов, называемых XML-документами. Если HTML является SGML-приложением, т. е. языком, определенным с помощью SGML, то XML является подмножеством SGML, т. е. упрощенным вариантом сложного языка SGML, созданным специально для использования в Интернет. XML полностью совместим с SGML. Основное назначение XML — структурирование информации, а ее форматирование при отображениях выполняется с помощью листов стилей, применяемых и в HTML. Язык XML используется в качестве средства для описания грамматики класса документов и контроля за правильностью разметки документов. Иначе говоря, сам по себе XML не содержит тегов, предназначенных для разметки, он просто определяет правила их создания. При этом совершенно нет необходимости отказываться от HTML. Язык XML просто дополняет язык HTML. Без знания HTML невозможно изучить и эффективно применять средства XML. Но XML позволяет расширять возможности языка разметки. Таким образом, если, например, для обозначения в документе элемента **rose** (роза) автор хочет использовать новый тег **<flower>** (цветок), то XML позволяет свободно ввести и использовать такой тег и в документ можно включать фрагменты, подобные следующему:

```
<flower>rose</flower>
```

(К сожалению, объяснение того, как автор добавил тег **<flower>** в используемый им язык разметки, выходит за рамки данного обзорного пособия, цель которого лишь познакомить читателя с общими направлениями развития Web-средств.)

Набор тегов может быть и далее расширен. Если, предположим, необходимо указать, что описание цветка по смыслу должно располагаться внутри описания оранжереи (conservatory), в которой он цветет, то в этом случае просто вводятся новые теги и определяется порядок их размещения относительно тегов **<flower>**, **</flower>**. После этого в документе может быть использована такая конструкция:

```
<conservatory>
<flower>rose</flower>
</conservatory>
```

Если нужно «посадить» туда (в оранжерею) еще несколько цветочков, то можно внести следующие изменения, смысл которых вполне ясен читателю, знакомому с HTML:

```
<conservatory>
<flower>rose</flower>
<flower>tulip</flower>
<flower>cactus</flower>
</conservatory>
```

Как видно, сам процесс разметки текста в XML-документе очень прост и требует наличия лишь базовых знаний HTML и понимания тех задач, которые необходимо выполнить, используя XML в качестве языка разметки. С появлением XML у разработчиков Web-документов появляется уникальная возможность определять собственные команды, позволяющие им наиболее эффективно структурировать данные, содержащиеся в документе. Автор документа разрабатывает его структуру и строит необходимые связи между элементами, используя те команды, которые удовлетворяют его требованиям, а также задачам последующей информационной обработки, включающей выполнение операций просмотра, поиска и анализа документов.

Еще одним из очевидных достоинств XML является возможность использования его в качестве универсального языка запросов к хранилищам информации. Сегодня в глубинах W3C находится на рассмотрении рабочий вариант стандарта XML-QL (или XQL), который, возможно, в будущем составит серьезную конкуренцию SQL. Кроме того, XML-документы могут выступать в качестве уникального способа хранения данных, который включает в себя одновременно как средства для анализа информации, так и средства представления ее на стороне клиента. XML позволяет также осуществлять контроль за корректностью данных, хранящихся в документах, производить проверки иерархических соотношений внутри документа и устанавливать единый стандарт структуры документов, содержимым которых могут быть самые разные данные. Это означает, что его можно использовать при построении сложных информационных систем, в которых очень важным является вопрос обмена информацией между различными приложениями, работающими в одной системе. В самом начале работы над проектом, создавая на основе XML структуру механизма обмена информации, разработчик может избавить себя в будущем от многих проблем, связанных с несовместимостью форматов данных, используемых различными подсистемами проектируемой системы.

Еще одним из достоинств XML является то, что программы-обработчики XML-документов несложны. Уже сегодня появились и свободно распространяются всевозможные программные продукты, предназначенные для работы с XML-документами. XML поддерживается сегодня в Microsoft Internet Explorer 4 и IE5. Было заявлено о его поддержке во многих других продуктах. Все это дает основания предполагать, что, скорее всего,

в ближайшем будущем XML станет основным языком обмена для информационных систем, заменив собой HTML. На основе XML уже сегодня созданы такие известные специализированные языки разметки, как SMIL, CDF, MathML, XSL. Список рабочих проектов новых языков, находящихся на рассмотрении W3C, постоянно пополняется.

Язык XML сейчас поддерживается всеми приложениями MS-Office 2000. Если вы сохраняете документ как Web-документ из любой программы MS-Office 2000, то он создается на базе XML. Пользователь MS-Office может даже не догадываться об этом. Но наш читатель уже подготовлен к осознанию этого факта. Поэтому рассмотрим хотя бы на очень простом примере, как простейший документ, подготовленный в редакторе Word 2000, сохраняется в виде HTML-документа с применением средств XML.

Рассмотрим следующий пример. Напишем в Microsoft Word 2000 всего одно слово «привет». Затем заполним форму «Свойства» из меню «Файл» (как показано на рис. 17) и сохраним документ как Web-страницу. Рассмотрим теперь HTML-код этой страницы:

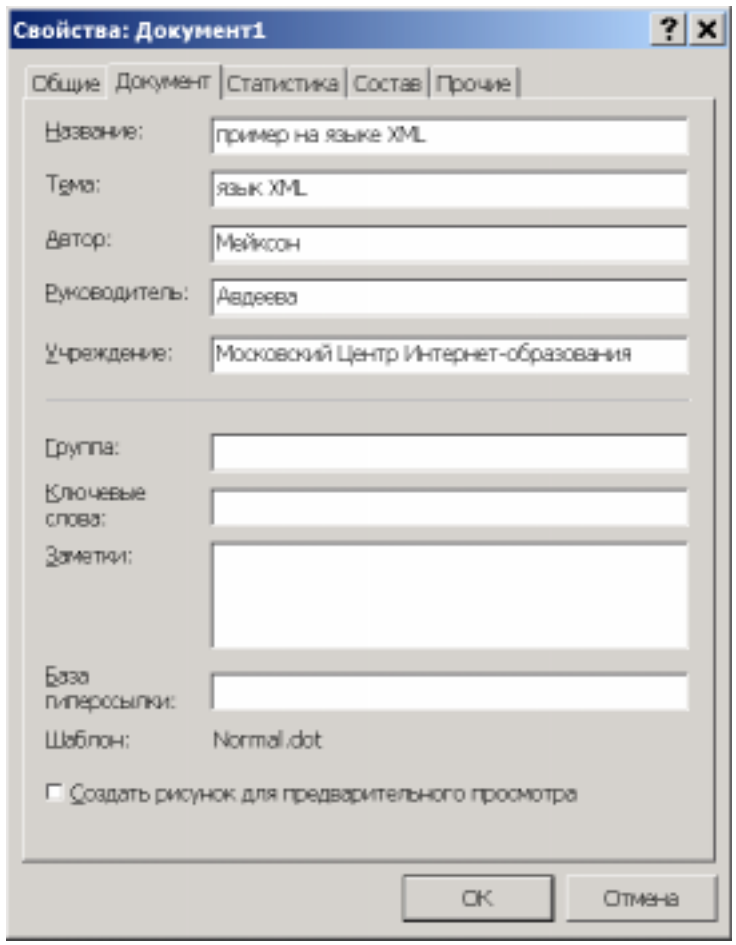

Рис. 17. Исходная информация для сохранения документа в MS Word 2000

```
<html xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns="http://www.w3.org/TR/REC-html40">
<head>
<meta http-equiv=Content-Type content="text/html; charset=windows-1251">
<meta name=ProgId content=Word.Document>
<meta name=Generator content="Microsoft Word 9">
<meta name=Originator content="Microsoft Word 9">
<link rel=File-List href="files/filelist.xml">
<title>пример на языке XML</title>
<!- -[if gte mso 9]><xml>
<o:DocumentProperties>
<o:Subject>язык XML</o:Subject>
<o:Author>Мейксон</o:Author>
<o:LastAuthor>Meykson</o:LastAuthor>
<o:Revision>1</o:Revision>
<o:TotalTime>6</o:TotalTime>
<o:Created>2000-10-08T11:20:00Z</o:Created>
<o:LastSaved>2000-10-08T11:31:00Z</o:LastSaved>
<o:Pages>1</o:Pages>
<o:Words>1</o:Words>
<o:Characters>6</o:Characters>
<o:Category>авторы книги</o:Category>
<o:Manager>Авдеева</o:Manager>
<o:Company>Московский Центр Интернет-образования</o:Company>
<o:Lines>1</o:Lines>
<o:Paragraphs>1</o:Paragraphs>
<o:CharactersWithSpaces>6</o:CharactersWithSpaces>
<o:Version>9.2812</o:Version>
</o:DocumentProperties>
</xml><![endif]- -> <!- -[if gte mso 9]><xml>
<w:WordDocument> <w:DisplayHorizontalDrawingGridEvery>0
</w:DisplayHorizontalDrawingGridEvery>
<w:DisplayVerticalDrawingGridEvery>0
</w:DisplayVerticalDrawingGridEvery>
<w:UseMarginsForDrawingGridOrigin/>
<w:Compatibility>
<w:FootnoteLayoutLikeWW8/>
<w:ShapeLayoutLikeWW8/>
<w:AlignTablesRowByRow/>
<w:ForgetLastTabAlignment/>
<w:LayoutRawTableWidth/>
<w:LayoutTableRowsApart/>
</w:Compatibility>
</w:WordDocument>
</xml><![endif]- ->
<style>
<!- -
```

```
/* Style Definitions */
p.MsoNormal, li.MsoNormal, div.MsoNormal
{mso-style-parent:"";margin:0cm; margin-bottom:.0001pt;
msopagination:widow-orphan; font-size:12.0pt; font-family:"Courier New"; msofareast-
font-family:"Times New Roman"; mso-ansi-language:RU;}
@page Section1{size:595.3pt 841.9pt; margin:3.0cm 89.85pt 3.0cm 89.85pt;
mso-header-margin:36.0pt; mso-footer-margin:36.0pt; mso-paper-source:0;}
div.Section1{page:Section1;}
- ->
</style>
</head>
<body lang=EN-US style='tab-interval:36.0pt'>
<div class=Section1>
<p class=MsoNormal style='text-indent:1.0cm'><span lang=RU style='font-size:
12.0pt;mso-bidi-font-size:9.0pt;font-family:"Times New Roman"'>привет<o:p></o:p></
span></p>
</div>
</body>
</html>
```

К сожалению, для полного понимания приведенного HTML-кода необходимо не только знание HTML-тегов, их атрибутов, но и знакомство с синтаксисом XML, чего мы не предполагаем у читателя. Дадим лишь самые общие пояснения. Код начинается со следующего тега:

```
<html xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns="http://www.w3.org/TR/REC-html40">
```

Здесь (в начальном теге элемента **html**) использованы специфические для XML атрибуты **xmlns:o, xmlns:w, xmlns,** подтверждающие «происхождение» документа (из Office, из Word), и адрес (URL) версии HTML, которой соответствует документ.

Затем следует огромный элемент **head**. В него вложены мета-теги (см. выше соответствующий раздел), элемент **title**, описание структуры документа на XML и лист стилей, определяющих его отображение. И тексты на XML и описания стилей заключены в скобки <!- -...- ->, т. е. являются комментариями в тексте HTML. Каждый XML-элемент вводится парой тегов <xml>...</xml>. Лист стилей входит в элемент **style**. Тело документа (элемент **body**) внешне мало отличается от тех HTML-текстов, которые приводились в предшествующих разделах пособия. К сожалению, подробное рассмотрение отличий выходит за рамки нашего пособия. Но самое главное нужно отметить. Обычный браузер, не понимающий XML, воспроизведет наш документ без всяких затруднений, так как все XML-элементы заключены в скобки комментариев.

Приведем более простой пример XML-документа. В этом примере описывается расположение новосибирских университетов. Здесь указывается, например, что Новосибирский

государственный университет расположен в городе Новосибирске, который, в свою очередь, находится в России. Для этого используется вложенность элементов XML:

```
<country id="Russia">
<cities-list>
<city>
<title>Новосибирск</title>
<state>Siberia</state>
<universities-list>
<university id="1">
<title>Новосибирский государственный технический университет</title>
<noprivate/>
<address URL="www.nstu.ru"/>
<description>очень хороший институт</description>
</university>
<university id="2">
<title>Новосибирский государственный университет</title>
<noprivate/>
<address URL="www.nsu.ru"/>
<description>тоже неплохой</description>
</university>
</universities-list>
</city>
</cities-list>
</country>
```

При поиске в этом документе анализирующая программа будет опираться на информацию о его структуре, определенной элементами разметки документа. Если, к примеру, требуется найти нужный университет в нужном городе, то следует просмотреть содержимое элементов <university>, находящихся внутри только одного конкретного элемента <city>. Поиск при этом, естественно, будет гораздо более эффективен, чем нахождение нужной последовательности по всему документу.

Заключение

Подведем краткий итог нашему обзору элементов разметки Web-страниц. Простота языка HTML, возможность готовить с его помощью Web-документы, используя простейшие текстовые редакторы, ориентация на язык HTML большинства более «изысканных» механизмов оформления Web-документов сделали его на сегодняшний день ядром всех технологий Web-системы. Наиболее эффективными средствами оформления внешнего вида отображения HTML-документа являются листы каскадных стилей (CSS) и вводимые с помощью стилей слои (Layers).

META-теги позволяют ввести набор ключевых слов, характеризующих содержание документа и применяемых для его индексации при использовании информационно-поисковых систем, а также определить многие параметры обработки документа браузером.

Формы являются удобным средством обеспечения интерактивности документа, но их применение требует навыков программирования.

Технология Server Side Include (SSI) позволяет избежать повторения одинаковых элементов в разных HTML-страницах одного Web-сайта.

Java-аплеты, скрипты в HTML-документах и динамический HTML дают возможность превратить HTML-документ фактически в программу, выполняющуюся в браузере и реагирующую на действия пользователей.

Технология активных серверных страниц ASP и конкурирующая с нею технология PHP дают еще одну возможность обработки на стороне Web-сервера информации, получаемой от пользователя. Это позволяет на стороне Web-сервера формировать HTML-документ с включением в него данных, соответствующих запросу пользователя.

Внедрение объектов в HTML-документ позволяет неограниченно расширять возможности документов, к примеру, вставлять в них мультимедийные компоненты (например, Flash-элементы).

Язык HTML не позволяет строго определять структуру документов, поэтому ее проверка на стороне клиена невозможна. Неверно структурированные документы браузер отображает непредсказуемым образом, при этом часто не сообщая пользователю о возникших затруднениях! Язык XML позволяет создавать строго структурированные документы, удобные для последующей обработки. Автор документа может расширять набор элементов (тегов), используемых для разметки XML-документов.

Учебное издание

Учебно-методический комплекс
«Интернет-технологии — образованию»

Павел Григорьевич Мейксон, Вадим Валериевич Подбельский

НАЧАЛА САЙТОСТРОЕНИЯ

Учебное пособие для системы дополнительного
профессионального образования

Зав. редакцией *А.И. Павлова*
Редактор-корректор *Н.В. Шерстенникова*
Компьютерная верстка *А.И. Попов*

Издание подготовлено и выпущено при участии
ЗАО «Гуманитарный издательский центр ВЛАДОС»

Федарация Интернет Образования

Россия, 113152, Москва, ул. Тульская, 59.
Тел. (095)247-28-80; факс (095)755-80-00.
http://www.fio.ru
E-mail: info@fio.ru, center@fio.ru

Лицензия ИД № 05141 от 22.06.2001 г.
Сдано в набор 05.11.01. Подписано в печать 12.03.02.
Формат 60×90/8. Печать офсетная. Бумага офсетная. Усл. печ. л. 6,5.
Тираж 5000 экз. Заказ № .

«Гуманитарный издательский центр ВЛАДОС».
117571, Москва, просп. Вернадского, 88.
Тел. 437-11-11, 437-25-52, 437-99-98; тел./факс 932-56-19.
E-mail: vlados@dol.ru
http://www.vlados.ru

ISBN 5-901891-02-3
9 785901 891025

ООО «Полиграфист».
160001, Россия, г. Вологда, ул. Челюскинцев, 3.