Лекция 1

Глава 1. Технологии и стандарты World Wide Web

Интерактивная мультимедийная гипертекстовая среда World Wide Web (WWW) появилась в современном виде благодаря наличию нескольких факторов. Первый из них – платформенно-независимый способ распространения информации.

Как известно, компьютеры совместно с операционными системами (DOS, семейство Windows, Windows NT, Mac OS, семейство UNIX, Linux и др.) образуют "*платформы*", на которых функционируют программы, реализующие разнообразные "*пользовательские среды*" (графические, текстовые, звуковые).

Для успешного взаимодействия в сети необходимы средства, не зависящие от конкретной компьютерной платформы. Такими средствами являются единое адресное пространство и сетевые протоколы.

Другая важная задача обеспечения эффективного взаимодействия компьютеров в сети связана с проблемой доступа в сеть, и, соответственно, с доступностью информации. Доступ в сеть обеспечивается не только с помощью компьютеров. Мобильные телефоны, карманные компьютеры, Web TV, продукты Apple (iPad, iPhone и т. д) обладают такой возможностью, но отличаются от персональных компьютеров размером экрана и устройствами ввода. Доступ в Internet различается также и в зависимости от способа представления информации, который отличен в текстовых, графических и речевых браузерах. Решение задачи связано с воплощением идеи разделения *структуры* и содержания документов от их *представления*. Представление может отличаться в разных устройствах доступа, тогда как структура и содержание остаются неизменными.

Следует также отметить совершенствование технологии "клиент-сервер", связанное с появлением графических многофункциональных браузеров, клиентских и серверных расширений, обеспечивающих эффективное интерактивное взаимодействие, а также совершенствование языков написания сценариев и серверных программ.

1.1. WWW

WWW образует *технологию обмена* гипертекстовыми документами. В ее основу положено использование идеи *гипертекста*, включающего в себя разнотипные данные, объединенные механизмом *гиперссылок*, позволяющим интегрировать документы, размещенные на разных компьютерах в разных сетях.

Проект WWW появился в начале 90-х годов и принадлежит Тиму Бернерс Ли, сотруднику Европейского института физики частиц (CERN).

Документы WWW называют также web-страницами, web-документами, html-файлами. Эти документы имеют стандартное расширение html или htm.

Основная особенность гипертекстовых документов – наличие гиперссылок, которые позволяют преодолеть ограничения на размер документа (hyper - сверх), а также н тип используемых данных. Таким образом, WWW можно представить в виде гигантской библиотеки гипертекстовых, интерактивных, мультимедийных документов, связанных гиперссылками.

В общем виде гиперссылка указывает на документ, находящийся на сервере и имеет следующие части:

http://www.server_name/directories/file_name

В первой части указан протокол, который использует web-сервис, затем имя сервера, директории, ведущие к файлу и имя целевого файла.

WWW использует для обмена данными протокол HTTP (HyperText Transfer Protocol, протокол обмена гипертекстом). Это высокоуровневый протокол (прикладного уровня), который работает «поверх» низкоуровнего протокола TCP/IP (Transfer Control Protocol/Internet Protocol, протокол управления обменом, протокол Internet).

Информационной единицей WWW является *сайт* (site – площадка) – площадка для размещения web-страниц. Сайт – это совокупность web-страниц, связанных одной темой, единым оформлением и имеющая имя.

Имя имеет следующий вид:

http://www.server_name/directories/index.html

После указания протокола следует доменное имя сервера, а затем, если необходимо, указываются папки и имя домашней страницы сайта. По умолчанию, домашняя страница имеет имя index.html или default.html.

Процесс размещения сайта на сервере является *публикацией* сайта. Поддержка работы сайта, серверных программ, а также баз данных, задействованных в работе сайта, называется *администрированием* сайта.

Область знаний, предметом которой является разработка (development) и представление (design) web-страниц называется *Web-дизайном*.

1.2. WWW Консорциум

В 1994 году был создана международная и некоммерческая организация Консорциум W3 (W3 Consortium, сокращенно W3C), занимающаяся разработкой стандартов для протоколов и языков, связанных с Web. Консорциум имеет официальный сайт в Internet – http://www.w3c.org/, где публикуется самая последняя информация, связанная со стандартами Web.

Можно выделить три основные цели этой организации:

- обеспечение универсального доступа к сети;

- разработка программных средств, позволяющих пользователям взаимодействовать в сети;
- управление развитием сети с учетом юридических, социальных или коммерческих аспектов.

Каждая новая версия спецификации любой сетевой технологии, прежде чем стать стандартом, подлежит тщательному рассмотрению в W3C.

1.3. SGML

В конце шестидесятых годов в фирме IBM был разработан язык разметки технической документации, который приобрел в 1986 году статус международного стандарта – SGML (Standard Generalized Markup Language). Этот обобщенный язык был предназначен для построения систем логической, *структурной* разметки любых разновидностей текста независимо от компьютерных платформ.

Структурная разметка означает, что управляющие коды, производящие такую разметку, не несут никакой информации о форматировании документа, а лишь указывают границы и соподчинение его основных частей, т.е. задают его *структуру*. Форматирование при этом выносится в так называемые стилевые спецификации – отдельный и допускающий независимые изменения информационный слой. Благодаря этому размеченный текст может интерпретировать любая программа, работающая с любым устройством вывода. Этот аспект тесно связан с обеспечением *доступности* информации в Internet не только для любых платформ, но и сред, которые не ограничиваются распространенной *графической* средой, а включают в себя и *текстовые* и *звуковые* среды.

1.4. Языки разметки HTML, XML, XHTML

Разметка web-документа осуществляется с помощью языков разметки. Разметка включает в себя описание структуры, в которую помещен content (содержание) документа. Внешний вид документа описывается с помощью самостоятельной технологии – каскадных таблиц стилей (Cascading Style Sheets, CSS).

HTML (Hyper Text MarkUp Language, язык разметки гипертекста) в течение некоторого времени существовал в версии 4.01. С ее стандартами можно познакомиться на сайте W3C.

HTML имеет один вид управляющих конструкций – теги (или дескрипторы). Они используются для описания структуры и для внедрения объектов на странице, таких как графические изображения, аудио-клипы или видео-ролики.

Элемент HTML состоит из следующих частей (на примере элемента **p** – параграфа):

```
<p align="center" id="para1">content</p>
```

открывающего тега с именем ** **, его атрибутов **align="center" id="para1"**, содержимого (content) и закрывающего тега ** **.

W3C стандартизировал набор логических тегов, которые должны заменить собой теги физического форматирования как не отвечающие требованиям разделения структуры и представления, а также универсальности. К примеру, тег физического форматирования **<b>** (bold), обеспечивающий полужирное начертание, не имеет смысла в речевом браузере, тогда как логический тег **** позволяет выделить контент в зависимости от типа среды, в которой он используется.

Традиционный HTML определяется шаблоном DTD (Document Type Definition, описание типа документа), который определяет набор тегов, их атрибуты, вложенность тегов и другие правила использования языка.

Набор тегов HTML фиксирован и ограничен, поэтому для HTML используется одно DTD, ссылка на которое выглядит следующим образом:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//En">

Сделать развитие языка естественным процессом, позволив браузеру обрабатывать теги, изобретенные разработчиком, можно используя XML (eXtansible MarkUp Language, расширяемый язык разметки) и его производные. Для браузеров мобильных телефонов аудиоподача информации, которая может осуществляться, например, синтезатором речи, выглядит вполне естественно. Синтезатор речи будет требовать какой-то определенный, особенный набор команд. Для мощных графических станций акценты делаются на трехмерную графику, для которой может быть введена своя система команд.

В настоящее время широко применяются, созданные на основе XML язык WML (Wireless MarkUp Language, язык беспроводной связи) для мобильной связи или MathML (язык разметки математических текстов) для передачи математических документов.

Языки, созданные на основе XML должны иметь соответствующие описания, которые помещаются в DTD или XML-схему. В отличие от HTML каждый язык должен обладать уникальным DTD. Оформление документов возможно только с использованием CSS или специального языка XSL (eXtansible Stylesheets Language, расширяемый язык стилевых спецификаций). Еще одной особенностью являются более строгие, чем в HTML синтаксические правила, что обеспечивает более простую машинную обработку.

Дальнейшее развитие языков разметки консорциум W3C связывал с созданием стандарта HTML нового поколения на основе XML (eXtansible MarkUp Language, расширяемый язык разметки).

В 2000 году W3C опубликовал рекомендации по использованию версии языка XHTML (eXtensible HTML, расширяемый HTML). По этому стандарту в код HTML вводится ряд ограничений, которые позволяют

рассматривать XHTML как подмножество XML. Благодаря этому XHTML оказывается совместим с другими существующими специализированными языками и обеспечивает преемственность HTML и XML.

Формальные изменения касаются более строгих правил разметки, которые устанавливают использование только парных тегов, нижнего регистра для записи тегов и атрибутов, заключение атрибутов в кавычки, а также выполнение правила вложенности тегов. Оформление документа возможно только с применением CSS.

В документе XHTML обязательны следующие теги:

<?xml version 1.0?> - объявляет документ документом XML

<!doctype…..> - ссылка на соответствующее DTD

<html xmlns=”…”> - ссылка на пространство имен

<head>….

Однако путь использования нового стандарта XHTML налагает обязательное требование жесткой обработки ошибок и новый MIME-тип, что делает невозможным использование большинства уже созданных web-страниц.

Поскольку консорциум закрыл рабочую группу HTML, сосредоточив усилия на совершенствовании спецификации XHTML, то дальнейшее развитие HTML было предпринято рабочей группой, не согласной с политикой W3C, которая получила название WHAT (Web Hypertext Application Technology Working Group).

Целью этой группы являлось разработка спецификаций на основе HTML и связанных с ним технологий. Основное внимание было сфокусировано на задаче документирования процесса разбора браузерами html-текста таким образом, чтобы обеспечить совместимость с существующим информационным наполнением Web без прекращения обработки документа и вывода сообщения об ошибке. Этот стандарт регламентирует и документирует множество функций, которые де-факто использовались web- браузерами годами.

Вместе с этим были разработаны спецификации, добавляющие новые элементы управления к формам HTML 4.01, спецификации, обеспечивающие встроенную поддержку для аудио и видео без использования сторонних подключаемых модулей, а также документированы новые функции для графического вывода и улучшения пользовательского интерфейса за счет новых "семантических" элементов разметки.

Расширенные возможности создания web-приложений и прикладные интерфейсы (API) Web Storage и Web SQL Database дают возможность хранить и обрабатывать данные на машине клиента.

Наработки группы WHAT были одобрены консорциумом W3C, и с 2010 года дальнейшая работа продолжалась под эгидой этой организации.

Новый набор самостоятельных функций получил название HTML5. В сочетании с технологией CSS3 HTML5 заложил основу для разработок web-приложений нового поколения.

В HTML5 существует только одно DTD, которое описывается тегом <!DOCTYPE HTML>.

В настоящее время спецификации HTML5 находятся в процессе разработки, однако формат HTML5 строится на базе HTML 4.01 и разработка ведется таким образом, чтобы обеспечить обратную совместимость с существующими браузерами.

В последних версиях браузеров Mozilla Firefox, Google Chrome, Safari, Opera, IE9 реализована поддержка HTML5.

1.5. Каскадные таблицы стилей (Cascading Style Sheets, CSS)

Язык HTML был создан для разметки научных текстов и не содержал первоначально средств оформления документов. С тех пор как Internet стал достоянием широкой общественности, появилась потребность в придании страницам гипертекста более привлекательного вида. Эта потребность побудила фирмы-производители браузеров создавать элементы HTML, назначением которых было форматирование документа. В силу коммерческих интересов были созданы многочисленные теги, которые корректно отображались только в браузерах производителей. В результате одна и та же web-страница выглядела по-разному в разных браузерах.

Для того чтобы вернуть HTML его истинное назначение и ликвидировать произвол в отображении документов W3C разработал технологию, независимую от HTML и предназначенную для *представления* документа. Эта технология получила название "Каскадные таблицы стилей (Cascading Style Sheets, CSS)". CSS первого уровня появились в 1995 году. В настоящее время созданы стандарты CSS2 и CSS3. Последние разработки браузеров поддерживают CSS2 и частично CSS3.

Важной частью стандарта является ввод понятия типа среды – текстовой, графической, звуковой, и конкретизация средств оформления страницы, соответствующих этим типам. Так, в звуковом типе среды определены свойства, позволяющие регулировать *громкость*, *темп* произнесения текста и *тембр* голоса. Таким образом, одна и та же структура и содержание документа по-разному представляется в зависимости от оформительских средств данного типа среды.

Из графических нововведений можно отметить возможность использовать *подключаемые шрифты*, относительное и абсолютное позиционирование объектов, а также использование пространственных эффектов.

Технология CSS формально независима от HTML, имеет совершенно иной синтаксис и позволяет задавать параметры графического (также как и текстового, и звукового) представления дескрипторов HTML.

Достоинствами CSS можно назвать:

- обеспечение возможности представления документа способом, независимым от описания структуры;
- гибкость, позволяющая при внесении изменений в файле описания стилевых свойств, связанного со страницами сайта, синхронно внести изменения во все связанные страницы;
- значительное расширение функциональных возможностей оформления контента.

Подключение стилей к документу может быть произведено тремя способами: встраиванием – описание стиля включается как атрибут к тегу HTML и действует только на этот тег; внедрением – описание стилей вынесено в заголовок документа и относится ко всей странице, и, наконец, связыванием, при котором стили определяются в отдельном файле, и этот файл связывается со страницами сайта, тем самым воздействуя на весь сайт.

При этом установлен приоритет стилей в порядке убывания: встраивание - внедрение – связывание. Приоритетностью стилей объясняется название стилей «каскадные».

1.6. JavaScript

Появление языков сценариев, интерпретируемых браузером, связано с востребованностью интеллекта не только на сервере, но и на клиентской стороне.

Скриптовой язык JavaScript поддерживался уже первым браузером, выпущенным фирмой Netscape. Он преобразовался из языка, имевшего первоначальное название LiveScript, и предназначен для управления содержимым страницы и собственно браузером.

Такие задачи как проверка данных формы перед ее отправкой на сервер, помещение страниц в окна с произвольным расположением и размером, создание меню и простой анимации решаются с помощью сценарного языка на клиентской стороне. В web-страницах используются языки JavaScript, получивший статус стандартного, и VBScript, который является разработкой фирмы Microsoft и поддерживается браузером Internet Explorer.

Язык JavaScript является интерпретируемым (отсюда название «язык сценариев» в отличие от языков программирования, которые являются компилируемыми) языком, ориентированным на обработку событий. Он относится к группе объектно-ориентированных языков (ООЯ) и строится на работе с объектами.

JavaScript в настоящее время широко используется и на серверной стороне. В частности, в составе активных серверных страниц (ASP, Active Server Pages) содержится html-код и фрагменты, написанные на скриптовых

языках. После выполнения кода таких страниц клиенту отсылается остаток в виде html-кода, который и воспроизводится браузером.

Язык JavaScript в сочетании с объектной моделью документа (Document Object Model, DOM) и каскадными таблицами стилей позволяет управлять элементами страницы и их свойствами после загрузки страницы в браузер. Эту комбинацию Web-технологий, основанную на различных стандартах (CSS, DOM, JavaScript и языков разметки) принято называть DHTML (Dynamic HTML, динамический HTML). DHTML позволяет существенно увеличить интерактивность Web-страницы. Созданная на основе DHTML страница может изменяться, не обращаясь к серверу за дополнительными данными.

Стандартизацией языка JavaScript занимается ECMA (European Computer Manufacturers Association). Его "официальная" версия называется ECMAScript**.**

В настоящее время созданы библиотеки JavaScript, которые содержат большое количество модулей (плагинов), предназначенных для создания элементов пользовательских интерфейсов, создания визуальных эффектов, а также манипулирования элементами web-страниц. Наиболее популярными являются библиотеки jQuery, YUI и Dojo.

1.7. Объектная модель документа

W3C разработал стандарт "объектной модели документа" (Document Object Model, DOM).

С помощью объектной модели документа (Document Object Model, DOM) можно менять web-страницу, используя язык написания сценариев.

DOM представляет собой программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому html-документов, и, соответственно, изменять содержимое, структуру и оформление таких документов.

DOM ставит в соответствие каждому элементу определенный идентификатор - атрибут ID. Идентификация позволяет элементу получить свой собственный уникальный адрес, а также преобразовать элемент в объект. Определяя элемент как объект, можно менять любое его свойство, заданное с помощью CSS или атрибута, а также сам элемент или даже его контент. Последняя версия стандарта DOM предусматривает мощный событийный интерфейс.

1.8. Клиент – серверная технология

Клиент-сервер — это вычислительная архитектура, в которой задания распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это вариант реализации программного обеспечения. Обычно эти программы расположены на разных вычислительных машинах и

взаимодействуют между собой через компьютерную сеть посредством сетевых протоколов, но их можно расположить также и на одной машине.

Логика взаимодействия и функциональное назначение программно-аппаратных комплексов в сети определяется в терминах технологии "клиент–сервер" (рис. 1.1).

Можно выделить три составляющие этой технологии:

- сервер, на котором находится информация;
- клиент, который формирует запросы на информацию и позволяет ее просматривать;
- протокол обмена, обеспечивающий их соединение и взаимодействие.

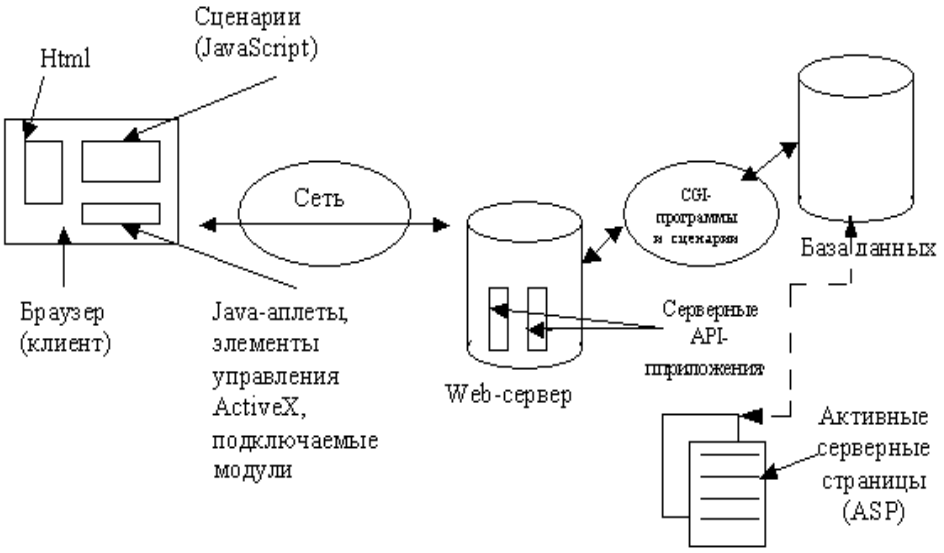


Рис. 1.1

1.8.1. Клиентская часть. Браузеры

Клиентской программой WWW является браузер (to browse – просматривать). Основная задача браузера – отображение web-страниц. Однако современные браузеры представляют собой интегрированные пакеты программ, имеющие более широкие возможности. Кроме основной функции браузер имеет возможность осуществлять поиск, сохранять закладки, кэшировать страницы, т.е. помещать их в особую папку на жестком диске с тем, чтобы иметь возможность просматривать их в автономном режиме (offline mode).

Первый графический браузер Mosaic появился в начале 90-х годов, затем фирма Netscape выпустила свой браузер, который в течение нескольких лет был единственным и достаточно функциональным программным продуктом. Но с появлением в середине 90-х Internet Explorer разработки Microsoft именно этот браузер занял лидирующее положение. Он установлен на всех компьютерах, работающих под управлением

Windows, является бесплатной, мощной, быстрой, весьма не требовательной к ресурсам и удобной программой.

По рейтингам 2016 года наиболее популярным браузером стал Google Chrome – один из самых используемых браузеров, который является безопасным и высокоскоростным браузером с удобным интерфейсом и стабильной и надежной работой.

Yandex.Browser – популярный в русском сегменте браузер с уклоном на сервисы одноименной компании. Он годится для использования на ПК, телефонах, планшетах. Он за короткое время завоевал популярность практически на всей территории России и стран СНГ.

Еще один браузер, набравший популярность в настоящее время – это Mozilla Firefox. Он является бесплатным и его программные коды открыты для изучения и модификации. Он быстр, поддерживает требования W3C.

Это второй по популярности в мире браузер и один из первых конкурентов Internet Explorer.

Браузер K-Meleon является одним из родственников браузера Mozilla Firefox. Он отличантся экономным использованием ресурсов компьютера.

Востребованной разработкой является норвежский браузер Opera. Эта мощная, компактная и очень быстрая программа поддерживает последние стандарты W3C и весьма требовательна к системным ресурсам. Новые усовершенствованные версии Opera появляются довольно часто.

Браузер Vivaldi продолжает и совершенствует традиции Opera – он быстрый, с богатой функциональностью, гибкий в настройке и ориентирован на индивидуальные требования пользователя.

Фирма Apple выпустила собственный браузер Safari. Утверждается, что это самый быстрый браузер и ожидается его модификация для Windows.

Для мобильных устройств широко применяется браузер Android фирмы Google.

Последние новости о браузерах можно прочитать, например, на сайте http://www.upsdell.com/Browser News/.

В качестве клиентов для обращения к веб-серверам кроме браузера могут использоваться разнообразные программы, самостоятельно обращающиеся к веб-серверам для получения обновлений или другой информации (например, антивирус может периодически запрашивать у определённого веб-сервера обновления своих баз данных), мобильный телефон, получающий доступ к ресурсам веб-сервера при помощи протокола WAP (Wireless Application Protocol), другие цифровые устройства или бытовая техника (Internet of Things - «интернет вещей»).

1.8.2. Клиентские расширения

К клиентским расширениям относятся подключаемые модули, элементы управления ActiveX фирмы Microsoft и Java аплеты – небольшие фрагменты на языке Java. Цель использования клиентских расширений одна – увеличение функциональности браузера.

Модуль (плагин) — это программный блок, который встраивается в браузер и расширяет его возможности. В отличие от дополнений модуль, как правило, не имеет интерфейса. Модули используются для проигрывания видео и аудио в браузере, просмотра PDF-документов, улучшения работы веб-служб, организующих совместную работу в интернете и т. д.

Некоторые типы мультимедийных данных поддерживаются браузерами непосредственно, например, это относится к графическим изображениям. Но охватить все форматы данных невозможно, поэтому и используются дополнительные программы, которые загружаются с сервера в случае их отсутствия на компьютере пользователя. Проверка наличия необходимой программы обработки данных осуществляется по т.н. MIME-типу (Multipurpose Internet Mail Extensions, многоцелевые расширения почты Internet), который определяется по формату передаваемого файла и прописывается в реестре Windows.

Получив от сервера в заголовке HTTP информацию о MIME-типе, браузер ищет программу, которая ассоциирована с данным типом. Если она не обнаруживается, то ее необходимо инсталлировать с сервера. В настоящее время многие из наиболее часто запрашиваемых модулей устанавливаются в браузерах в качестве стандартных функций, например, модуль Flash. При установке Windows некоторые программы устанавливаются автоматически, например, Microsoft Media Player.

ActiveX — фреймворк для определения программных компонентов, пригодных к использованию из программ, написанных на разных языках программирования. Программное обеспечение может собираться из одного или более таких компонентов, чтобы использовать их функциональность.

Технология ActiveX работает только на платформе Windows и служит тем же целям, что и подключаемые модули. Ее недостатком является низкая безопасность, которая должна быть компенсирована использованием электронных подписей, удостоверяющих, что фрагмент программного кода, который попадает на компьютер пользователя для воспроизведения, действительно принадлежит разработчикам ActiveX.

Java-аплеты представляют собой фрагменты на языке Java, реализующий небольшой законченный алгоритм. Их преимуществом является независимость от платформы, свойственная Java. В настоящее время Java-аплеты применяются достаточно

1.8.3. Серверная часть

Программы — сервера ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством протоколов HTTP, FTP, BitTorrent или потоковое мультимедиа, или работа с базами данных) или сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями, просмотр web-страниц). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, ей может обычно требуется высокопроизводительный компьютер. Из-за особой роли этого компьютера в сети, специфики её оборудования и программного обеспечения её так же называют сервером.

Web-сервер представляет собой совокупность аппаратных и программных средств, обеспечивающих взаимодействие с клиентом. Аппаратные средства должны содержать быстрые жесткие диски, значительный объем оперативной памяти для поддержания многочисленных отдельных процессов и скоростной сетевые интерфейсы (зависящие от сетевых карт). Основной функцией сервера является формирование ответов на HTTP-запросы, поступающие от клиента. Серверы могут либо напрямую возвращать файлы HTML, изображения, мультимедийные файлы, либо запускать программы или сценарии, которые возвращают аналогичный результат.

Все серверы обрабатывают транзакции HTTP, но все они обладают средствами, позволяющими не ограничивать их функциональность передачей файлов. К таким средствам относятся базовые службы безопасности, поддержка протокола HTTPS для обеспечения защищённых соединений с клиентами, ведение журнала обращений пользователей к ресурсам, аутентификация и авторизация пользователей, поддержка динамически генерируемых страниц, а также средства выполнения программ и сценариев (серверные приложения).

Часто на компьютере вместе с веб-сервером устанавливается также и почтовый сервер.

В последние годы наиболее распространённым web-сервером, занимавшим более половины рынка, являлся Apache — свободно распространяемый, высокоскоростной web-сервер, имеющий множество бесплатных модулей, выполняющих сжатие текста, проверку орфографии и других. Сервер наиболее часто используется в UNIX-подобных операционных системах.

К другим известным web-серверам относятся следующие:

IIS от компании Microsoft, распространяемый с ОС семейства Windows. Его преимуществами является интеграция с другими продуктами Microsoft, наличием платформы для разработки активных серверных страниц (ASP), а также бесплатная поставка вместе с операционной системой.

Сервер nginx (engine-x) — высокопроизводительный веб-сервер (разработанный Игорем Сысоевым) и пользующийся большой популярностью на крупных сайтах. nginx — простой, быстрый и надёжный сервер, не перегруженный дополнительными функциями.

Сервер lighttpd — свободный веб-сервер, работающий в Linux и других Unix-подобных операционных системах, а также в Microsoft Windows. Он позволяет использовать приложения, написанные на любых языках программирования

Google Web Server — веб-сервер, основанный на Apache и доработанный компанией Google.

Cherokee — свободный веб-сервер, управляемый через web-интерфейс.

Rootage — веб-сервер, написанный на языке Java.

THTTPD — простой, небольшой, быстрый и безопасный веб-сервер для Unix--подобных операционных системах, таких как FreeBSD, SunOS 4, Solaris 2, BSD/OS, Linux, OSF.

1.9.4 Серверные программы

Web-серверы по запросу пользователя могут возвращать различные файловые объекты (HTML-документы, изображения, мультимедийные файлы и т.д.), либо запускать программы или сценарии, которые предварительно обрабатывают информацию, полученную от пользователя и учитывают ее в конечном результате или информацию из баз данных.

В первом случае сайт позиционируется как статичный.

Во втором случае происходит динамическая генерация страниц сайта на основе результата обработки серверной программой данных, введенных пользователем в формы или на основе информации, хранящейся в базе данных.

Серверные программы обрабатывают данные пользователя и возвращают результат в виде HTML-файлов, которые традиционно интерпретируется браузером. Серверные программы не имеют интерфейса с пользователем и получают данные, введенные пользователем, от сервера, ему же передается и результат их обработки для дальнейшей передачи пользователю.

Серверные программы делятся на следующие виды.

1. Исполняемые программы, работающие через интерфейс CGI (Common Gateway Interface, общий интерфейс обмена), могут создаваться на любом языке программирования и представляют собой обычные откомпилированные файлы. При поступлении на сервер нескольких запросов на обработку данных, он запускает соответствующее количество копий CGI-программы. Достоинством таких программ является легкость создания (использование систем программирования типа Borland Delphi) и простота отладки, а недостатком является большой расход системных

ресурсов. Поскольку CGI-приложения представляют собой отдельные программы, то они выполняются не в адресном пространстве сервера и не могут внести сбой в его работу.

К примеру, обратиться к исполняемой программе можно, указав в атрибуте action следующий адрес: http://www.somesite.ru/cgi-bin/program.exe.

2. Активные серверные страницы (ASP, JSP и др.) представляют собой Web-страницы, которые кроме обычного HTML - кода, включают в себя команды языка сценариев. На стороне сервера эти сценарии исполняются, и пользователь получает остаток в виде привычного для браузера HTML–кода. В качестве сценарных языков используются VBScript и JavaScript в серверных страницах ASP, поддерживаемых Microsoft Internet Explorer, ColdFusion от Macromedia. В последнее время популярна технология создания серверных страниц PHP (Personal Home Page, персональная домашняя страница). Программный код пишется на языке PHP. Интерпретатор, выполняющий код PHP, распространяется бесплатно с открытыми исходными текстами, что и обусловило популярность данной технологии.

Языки сценариев на стороне сервера часто применяются для создания динамических страниц из баз данных, персонализации, генерирования на страницах компонентов, годных для повторного использования. Достоинством активных серверных страниц является легкость написания кода, а недостатком - их медленная работа в случае объемных сайтов.

К примеру, обратиться к активной серверной странице можно, указав в атрибуте action следующий адрес:

http://www.somesite.ru/asps/active_page.asp.

В настоящее время достаточно широко используется усовершенствованная технология ASP – ASP.NET, которая принципиально отличается тем, что позволяет разработчикам использовать практически любые языки *программирования*, в том числе входящие в комплект .NET Framework (C#, Visual Basic.NET, Jscript.NET). ASP.NET имеет преимущество в скорости по сравнению со скриптовыми технологиями, поскольку использует скомпилированный код.

3. Серверные сценарии подобны активным серверным страницам тем, что являются интерпретируемыми, но представляют собой только программный код без фрагментов HTML. Они обычно создаются на интерпретируемых языках (Perl, Python, VBScript, JavaScript), работают через интерфейс CGI и имеют те же недостатки, что и CGI - программы. Для обработки каждого набора данных пользователя запускается своя копия интерпретатора, который расходует много ресурсов на обработку сценария.

К примеру, обратиться к серверному сценарию можно, указав в атрибуте action следующий адрес:
http://www.somesite.ru/scripts/perl_script.pl.

4. Серверные расширения - приложения формата API (Application Programming Interfaces, интерфейсы программирования приложений) - это специальные программы, встроенные в Web - сервер в виде библиотек. Среди программ API - такие как ISAPI для сервера IIS от Microsoft, NSAPI для сервера от Netscape, Apache - модули для Apache. Преимуществом таких программ является их высокая производительность и малый расход системных ресурсов, а недостатком - сложность написания программ и возможность сбоя работы сервера в случае наличия ошибок в программе.

К примеру, обратиться к расширению Web-сервера можно, указав в атрибуте action следующий адрес: http://www.somesite.ru/bin/extension.dll.

Из современных серверных технологий следует отметить технологии корпорации Microsoft, которые получили название AJAX (Asynchronous Javascript and XML) – это подход к построению интерактивных пользовательских Web-приложений, заключающийся в фоновом обмене данными браузера и Web-сервера.

1.9.5 Взаимодействие клиента с сервером по протоколу HTTP

Запрос клиента осуществляется с помощью протокола прикладного уровня HTTP (yper Text Transfer Protocol, протокол передачи гипертекста), предназначенного для координации обмена данными между Web-сервером и браузером.

Процесс запроса web-страницы с участием протокола HTTP происходит следующим образом.

К примеру, если пользователь запрашивает документ с URL http://www/webdesign.com/examples/index.html, то для извлечения с сервера браузер сгенерирует запрос типа:

Get /examples/index.html HTTP/1.1

В этой строке запроса после указания метода Get, который используется в данном примере для передачи данных, следует относительный адрес домашней страницы и версия HTTP-протокола.

Кроме этих данных в запросе указывается тип браузера, используемый в нем язык, поддерживаемая кодировка страницы, а также поддерживаемые браузером MIME- типы.

MIME (Multipurpose Internet Mail Extensions, многоцелевые расширения почты Internet) – это стандарт, описывающий передачу различных типов данных по электронной почте, а в более широком понимании – спецификация для кодирования информации таким образом, чтобы ее можно было передавать по Интернету.

Get /products/index.html HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 2000)

Accept: application/x-comet, image/gif, image/jpeg, image/x-xbitmap, */*

Accept-Language: en

Accept-Charset: iso-88591

С помощью информации, содержащейся в запросе, можно определить окружение пользователя и настроить страницу в соответствии с типом браузера или используемым языком.

Далее от сервера следует ответ, содержащий соответствующий код. Например, если страница не существует, ответ имеет вид 404 Not Found. Если все идет хорошо, то ответ будет иметь вид

HTTP/1.1 200 ok

Кроме кода ответа заголовок HTTP содержит следующую дополнительную информацию:

Date:Mon, 30 Jan 2006 02:37:58 GMT

Server: Apache/1.3.4 (Unix)

Last Modified:Tue, 22 Dec 2005 21:04:18 GMT

Content-Length:7099

Connection:Close

Content-Type: text/html

Далее следует запрашиваемый документ:

<html>…</html>

Важной информацией заголовка является строка, содержащая тип передаваемого документа (MIME-тип) - Content-Type: text/html.

Тип MIME состоит из двух частей: обозначение типа/обозначение подтипа, например, text/html, image/gif или video/quicktime.

Когда браузер получает ответ, то по MIME-типу он определяет программу - обработчик передаваемых данных и передает ей управление (соответственно, это может быть подключаемый модуль, элемент управления, приложение и т.д.). В браузерах существуют специальные таблицы соответствия, по которым MIME-типы ассоциированы с обрабатывающими программами.

Далее взаимодействие продолжается следующим образом. Браузер читает доставленный html-код и лишь затем по очереди загружает объекты, находящиеся на web-странице. Это могут быть изображения, звуковые и видео-файлы, Flash-файлы или файлы приложений, например, Excel.

Когда браузер встречает объект на странице, он вновь посылает запрос серверу, указывая в заголовке имя файла, который должен быть загружен, например,

Get /images/logo.gif HTTP/1.1

Сервер ответит как и в предыдущем случае, но на этот раз с указанием MIME-типа image/gif, после чего будет передан код, формирующий изображение. Этот цикл повторяется до тех пор, пока запрашиваемая страница не будет передана полностью.

При создании web-страниц следует учитывать, что на время загрузки влияет не только такой фактор, как размер файла, но и количество запросов клиента, которые необходимо обработать web-сервером для полной загрузки страницы. А это, в свою очередь, определяется числом объектов, находящихся на web-странице.

Лекции 2,3

Глава 2. Технологии создания web-страниц. Язык разметки гипертекста HTML

2.1 Синтаксис

Синтаксис языка HTML (Hyper Text Markup Language) достаточно прост. Кроме обычного текста html-файл содержит только один тип управляющих конструкций, так называемые *дескрипторы* (или *теги*). Дескрипторы разделяют исходный текст на *элементы* и создают новые элементы, например, мультимедийные вставки.

Открывающий тег, контент (содержимое) и закрывающий тег образуют элемент.

Элементы бывают двух видов – *парные*, охватывающие какой-то фрагмент текста или (и) другие элементы и *непарные*. Например, парный элемент <**h1>** используется для создания заголовка первого уровня: **<h1>текст заголовка</h1>**, непарный элемент <**img>** используется для размещения на странице графического изображения: **<img src="URL">.**

Парные элементы должны вкладываться друг в друга без пересечений, т.е. если в области действия элемента A открылся элемент B, то элемент B должен закрыться до того, как закроется элемент A.

Многие элементы имеют *атрибуты*, которые уточняют и изменяют значение элемента. В большинстве случаев атрибуты не являются обязательными и интерпретатор HTML должен использовать значения по умолчанию. Атрибуты включаются в элементы в виде пары *атрибут="значение"* и отделяются друг от друга пробелами. Например, элемент **img** может иметь несколько атрибутов: **<img src="URL" alt="текст" width="значение" height="значение">**

Согласно стандарту, кавычки вокруг значения атрибута обязательны в тех случаях, когда значение содержит какие-либо символы, кроме букв, цифр, точки или дефиса. Регистр букв в идентификаторах элементов и атрибутов не имеет значения, но с точки зрения совместимости с XML лучше всегда пользоваться кавычками и использовать нижний регистр.

HTML-файл обычно имеет расширение **html**. Его код заключен в контейнерный элемент верхнего уровня **<html> </html>** В нем расположена область заголовка, образованная элементом <**head**> </**head**>, которая содержит служебную информацию о документе, сосредоточенную в элементах <**meta**>, информацию о стилях в элементе <**style**> и код JavaScript – в элементе <**script**>. Кроме того, в заголовке содержится заглавие страницы в элементе <**title**>, информация в котором, наряду с содержимым элементов <**meta**>, важна для поисковых роботов.

Контент страницы расположен в элементе <**body**> </**body**>, который расположен за элементом <**head**> </**head**> в контейнере <**html**> </**html**>.

Разметка текста происходит с помощью элементов, которые удобно расположить по группам в зависимости от их функционального назначения:

1. элементы *структуры* документа (заголовки шести уровней **<h1>…<h6>**, выделение блока, или слоя ** **, встроенный фрагмент ****, абзац **<p>**, цитата **<blockquote>**, перевод строки **
, **<nobr>, **<wbr>**, списки нумерованные ** …
**, списки маркированные ** …
, списки-определения **<dl>, <dt>, <dd>, вывод отформатированного текста **<pre>**, разделитель **<hr>**, информация об авторе **<address>**);
2. элементы *логического* форматирования (курсив **…**, полужирное начертание **…,** код программы моноширинным шрифтом **<code>…</code>,** выделение переменных в коде программы **<var>…</var>,** выделение текста, введенного с клавиатуры **<kbd>…</kbd>,** термины-определители **<dfn>…</dfn>,** цитаты **<cite>…</cite>,** аббревиатура **<abbr>…</abbr>**);
3. элементы *таблиц* (таблица **<table >…</table>,** заголовок таблицы**<caption>…</caption>,** заголовки столбцов вверху таблицы **<thead>…</thead>**, заголовки столбцов внизу таблицы **><tfoot>…</tfoot>**, тело таблицы **<tbody>…</tbody**, строка **<tr>…</tr>**, ячейка **<td>…</td>**, полужирное начертание в ячейке **<th>…</th>**, организация колонок <**col**> и <**colgroup**>…</**colgroup**>);
4. элемент *ссылок* (**<a>**);
5. элементы *форм* **(<form>…</form>, <input>,<select><option>…</option><select>);**
6. элементы *графических* объектов (**<img>);**
7. элементы *мультимедиа* HTML5 (**<video>,** <**audio**>);
8. семантические теги HTML5 **(<header>, <footer>, <section>, <article>, <nav>, <aside>, <figure>, <figcaption>;**
9. Для ввода в документ символов, отсутствующих на клавиатуре, или имеющих в синтаксисе HTML специальное значение, употребляются подстановки (entities) двух видов – мнемонические и числовые. Мнемонические подстановки имеют вид:

   &мнемонический код;

   например, для ввода знака **<** используется подстановка **<.**

В числовых подстановках используется десятичный числовой код нужного символа из стандарта **Unicode** с добавлением впереди символа **#.** Так, для ввода символа неразрывного пробела используется код ** .**

Семантические теги HTML5 предназначены для размещения информации определенного смыслового наполнения.

С помощью элементов <**header**> и <**footer**> размечается заголовочная часть страницы и «подвал» сайта или раздела, в нем обычно располагается имя автора, дата документа, контактная и правовая информация.

Элемент <**section**> определяет раздел документа, который может включать в себя заголовки, шапку, подвал и текст.

Элемент <**article**> включает в содержание сайта информацию в виде новости, статьи, записи блога, форума.

Элемент <**aside**> определяет блок, который не относится к основному контенту и служит для размещения рубрик, ссылок на архив, меток и другой информации. Такой блок, если он располагается сбоку, называется, как правило, «сайдбар» или «боковая панель».

Элемент <**nav**> используется для расположения навигационных элементов.

Элементы <**figure**>, <**figcaption**> используются для группирования любых элементов, например, изображений и подписей к ним.

В HTML 4.01 благодаря использованию CSS можно было создавать страницы с понятной для пользователей визуальной структурой, но эти страницы не были однозначно идентифицированы поисковыми роботами или браузерами.

Например, поисковый робот не может отличить содержимое документа от навигационного меню, если они размечены с помощью одинаковых **div**-элементов.

Для разрешения этой проблемы в HTML5 и были введены семантические тэги. С помощью семантических тэгов можно сделать страницы сайтов более понятными для поисковых систем и браузеров.

Рис. 2.1

Поисковые системы лучше индексируют сайт, поскольку чётко отделяют контент страницы от вспомогательных элементов. Речевые браузеры, предназначенные для слабовидящих людей, пропускают заголовок и переходят непосредственно к содержимому.

На рис. 2.1 показан пример структуры документа в HTML 4.01.

На рис. 2.2 показана структура документа в HTML5 с использованием новых семантических тегов разметки.

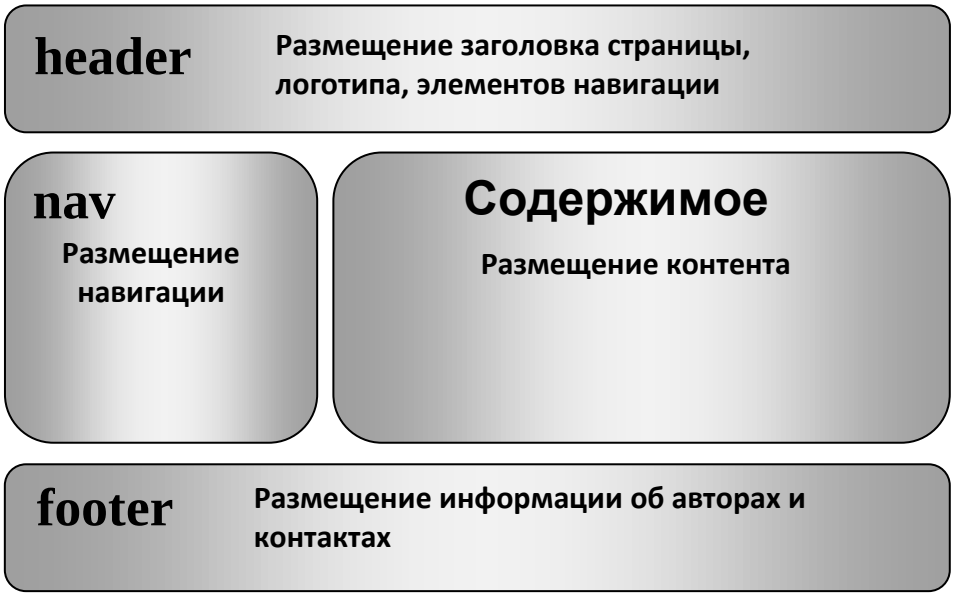


Рис. 2.2

2.2 Инструменты HTML

2.2.1 Текстовые редакторы

Информация HTML хранится в обычном текстовом формате, поэтому для кодирования html-документов можно пользоваться обычными текстовыми редакторами. В среде Windows простейшим редактором является Notepad (блокнот). Этот способ кодирования требует досконального знания HTML, однако, он лишен таких недостатков, как избыточность кода или его модификация, которые зачастую имеют место при использовании приложений WYSIWYG. Только при кодировании в текстовой среде разработчик имеет полный контроль над кодом страницы.

2.2.2 Специализированные редакторы

Специализированные текстовые редакторы занимают промежуточное место между простым текстовым редактором и редактором WYSIWYG.

Обычно в них используется простой графический интерфейс и большое количество мощных инструментов. Эти средства желательно применять, обладая значительными знаниями HTML, когда требуется

работать более эффективно. Процесс создания кода ускоряется за счет наличия таких инструментов, как, например, проверка орфографии.

В специализированных текстовых редакторах предлагаются шаблоны, инструментальные панели для вставки элементов, а также мастера изображений, которые автоматически вставляют в код их размеры.

Редактор Notepad++ текстовый редактор, который можно использовать как более функциональную замену Блокноту Windows, так и в качестве редактора с языками верстки и web-программирования: HTML, CSS, Java Script, PHP.

Работа в режиме вкладок, подсветка синтаксиса, кодировки, макросы, возможность установки дополнительных плагинов, проверка кода и сравнение файлов – эти и другие возможности делают Notepad++ действительно полезным инструментом для web-разработчика.

Стоит отметить легкость дистрибутива и скорость работы программы.

Из данной группы редакторов можно отметить HotDog Pro, HomeSite для Windows, Smultron для Mac и Bluefish для Linux.

2.2.3 WYSIWYG редакторы

Приложения данной группы также обладают многофункциональным графическим интерфейсом, позволяющим размещать необходимую графику и текст так, как эти элементы должны выглядеть на web-странице. За последующую генерацию кода отвечает программное обеспечение. Некоторые приложения WYSIWYG имеют мощные дополнительные возможности, не связанные с созданием страниц. К таким возможностям относятся расширения, позволяющие управлять проектами или специальные приложения для поиска и поддержки форм, как в программе FrontPage фирмы Microsoft.

Недостатками данных редакторов является их сложность, а также генерация избыточного кода. Кроме того, приложения WYSIWYG зачастую не успевают отслеживать те изменения, которые происходят в среде WWW. Появление новых форматов файлов, подключаемых модулей и функций браузеров требует своевременного обновления приложений WYSIWYG.

Самый популярный WYSIWYG-редактор HTML и других языков web-разработки – Adobe Dreamveawer. Он отличается удобным интерфейсом, большим количеством доступных инструментов, поддержкой актуальных технологий и интеграцией с другими приложениями пакета Creative Suite.

Большим плюсом Adobe Dreamveawer является наличие подробной профессиональной документации, а также множества бесплатных уроков по работе с программой, которые можно найти в сети Интернет. С помощью Adobe Dreamveawer можно создавать код на разных языках программирования и верстки – HTML, CSS, PHP и др.

Кроме того, он может работать с распространенными движками (CMS), такими как Joomla, WordPress и Drupal. Актуальной сейчас является версия CS6.

Еще один популярный визуальный редактор - WYSIWYG Web Builder, который недавно обновился до версии 9. Это мощное решение, с помощью которого можно создавать качественные web-сайты, не обладая при этом навыками верстки и web-программирования. Программа позволяет создавать как простые сайты-визитки или корпоративные ресурсы, так и многостраничные ресурсы со сложными сценариями и вставкой интерактивных элементов. В комплекте идут несколько различных шаблонов, а в процессе работы можно добавлять новые. В последней версии WYSIWYG Web Builder появился ленточный интерфейс, добавлены новые возможности для работы с CSS3 и больше сотни других улучшений.

Кроме коммерческих решений, есть и хорошие open-source решения. Например, бесплатный визуальный редактор BlueGriffon, созданный на движке Gecko, на котором работает также всемирно известный браузер Mozilla Firefox.

Это современное и надежное решение для редактирования web-страниц и создания сайтов, соответствующее последним стандартам Web. BlueGriffon является визуальным редактором HTML, PHP, CSS и других языков web-разработки, который обладает интуитивно понятным удобным интерфейсом, а редактируемый в нем документ будет выглядеть точно так же как в браузере Firefox.

WYSIWYG-редактор является действительно удобным инструментом для редактирования и web-страниц и создания полноценных сайтов разного уровня сложности. При этом, пользователь может создавать привлекательные web-сайты без необходимости обширных технических знаний.

Пользователи UNIX могут использовать пакет WebWorksPablisher фирмы Quadralay.

2.2.4 Шаблоны HTML-документа

Для ускорения работы по созданию web-страниц разработчик может использовать шаблоны, которые видоизменяются в зависимости от содержания страницы. Так, может быть создан шаблон, содержащий таблицы, списки и другие элементы HTML, необходимые разработчику. Удобно поместить его в папке с остальными файлами разработки и дать шаблону соответствующее имя, например, template.html.

Пример шаблона, содержащего основные части HTML-документа приведен ниже.

В структуре html-документа можно выделить следующие части, каждая из которых ограничена парными элементами.

Часть **<html>…….</html>,** внутри которой расположена область *заголовка*, ограниченная элементами **<head>…..</head>** и область *тела* документа между элементами **<body>…….</body>.** В области заголовка **<head>…..</head>** расположены область **<title>……</title>**, содержащая *название (заглавие)* страницы, отображаемое в окне браузера**,** область записи *сценария* **<script>……</script>,** а также область расположения *стилевых спецификаций* **<style>…….</style>.**

```
<html>
  <head>
    <title>
    </title>
    <script language="JavaScript" type="text/javascript">
    <!--
    ……
    //- ->
    </script>
    <style type="text/css">
    ……
    </style>
  </head>
<body>
……
</body>
</html>
```

В html-документе могут быть использованы *комментарии*, которые заключаются в элементы **<!--…-->.**

Далее приведен листинг html-документа, который использует элементы создания заголовка первого уровня **<h1>,** абзаца с правосторонним выравниванием **<p align="right">**, логического выделения **,** а также формирования нумерованных <ol> и маркированных **<ul>** списков и таблицы **<table>**. В качестве содержимого ячейки таблицы используется графическое изображение **<img>**, а элементами маркированного списка служат гиперссылки **<a>**.

```
<html>
<head>
<title>Пример HTML-кода</title>
<style type="text/css">
body
 {background-color:#c0c0c0;
```

```
  color: #000000;
  font-size:14pt}
p
  {text-align:right}
</style>
</head>
<body>
<h1>HTML-документ</h1>
<p>Абзац с правосторонним выравниванием</p>
<!—Использование логического выделения-->
<em> Нумерованный список: </em> <br>
<ol>
<li>пункт1</li>
<li>пункт2</li>
<li>пункт3</li>
</ol>
<em> Маркированный список: </em> <br>
<ul type="square">
<li><a href="file1.html">ссылка на файл1</a></li>
<li><a href="file2.html">ссылка на файл2</a></li>
<li><a href="file3.html">ссылка на файл3</a></li>
</ul>
<table border="2px">
<tr>
<td>cats</td>
<td><img src="cats.jpg" border="6px" ></td>
</tr>
<tr>
<td>telephon</td>
<td>1234567</td>
</tr>
</table>
</body>
</html>
```

Результат отображения кода в браузере приведен на рис. 2.3.

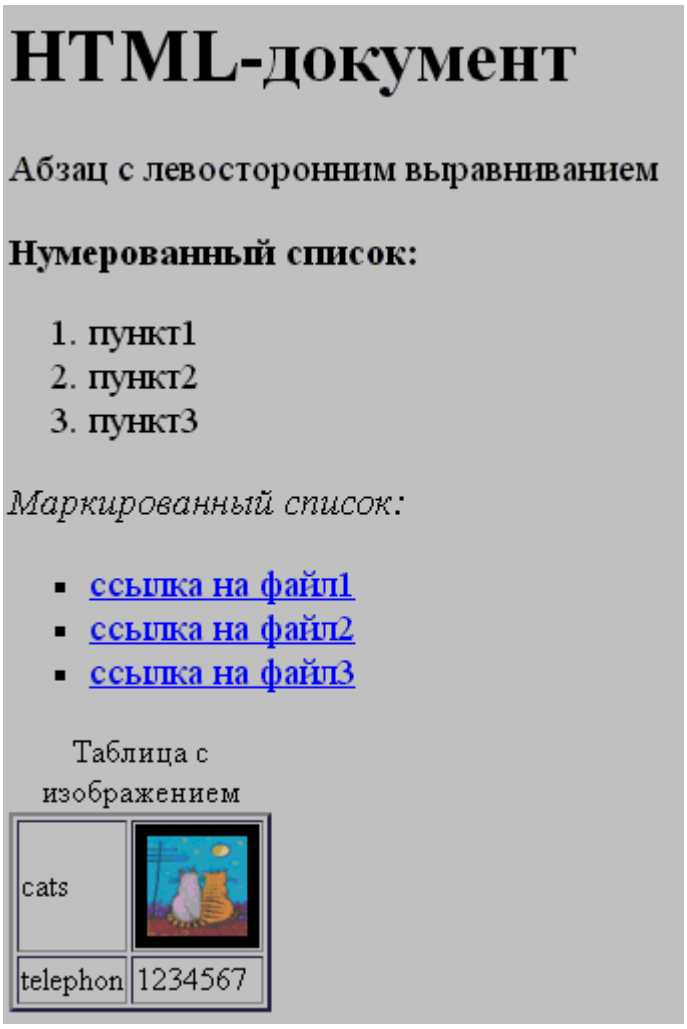


Рис. 2.3

2.3 Гиперссылки и навигация

Гиперссылки являются характерным элементом гипертекстовых документов. Рассмотрим элемент HTML, который обеспечивает формирование гиперссылок.

В общем виде гиперссылка на внешний (удаленный) файл представляется следующим образом:

http://www.server_name/directories/file_name#anh?param

Ее части - протокол, сервис, имя сервера, директории, ведущие к файлу, якорь, указывающий на фрагмент файла и список параметров, которые необходимо передать серверу, образуют **URL (Universal Resource Locator**, универсальный указатель ресурса).

2.3.1 Элемент anchor и его атрибуты

В HTML гиперссылки формируются с помощью элемента **anchor <a>…** и обязательного атрибута **href** (hyperreference). В качестве контента обычно используется текст:

**текст ссылки **

Часто применяются ссылки, в которых вместо текстового контента используется изображение:

<img src=”file_name.gif”…>

Кроме обязательного атрибута **href** для пояснения целевой функции гиперссылки применяется атрибут **title**, который формирует всплывающую подсказку при наведении мыши на гиперссылку (рис. 2.4)

**text1 **

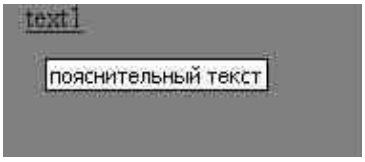


Рис. 2.4

Целевой файл может быть открыт в поименованном окне или в стандартных окнах, имеющих зарезервированные названия. В этих случаях используется атрибут **target:**

**текст ссылки **

Зарезервированными названиями для окон являются: **_top, _parent, _self, _blank**.

Для поддержки ссылки клавиатурой используется атрибут **accesskey**

**текст ссылки **

При этом сочетание клавиш alt+y активизирует ссылку.

Можно создавать гиперссылки не только на документы WWW, но и на файлы других сервисов Internet.

Ссылка на электронную почту:

Пишите мне

Ссылка на FTP:

Архив

Для ссылки на определенный фрагмент страницы используют установку якоря на этот фрагмент, а затем ссылку на якорь.

В пределах одной страницы часто ссылку делают в ее начало, особенно если страница занимает более одного экрана.

Установка якоря производится элементом <**a**> с атрибутом **name**:

начало текста, а в нижней части страницы устанавливается ссылка на якорь:

в начало текста

2.3.2 Относительные ссылки

Как известно, все файлы сайта должны находиться в одной корневой папке, которая размещается на сервере.

Внутри сайта используются относительные ссылки на файлы корневой папки.

Пример 1. Создать гиперссылку на файл **image1.gif** из файла **index.html**. Расположение файлов показано на рис. 2.5.

В данном случае гиперссылка будет иметь вид: **смотри рисунок**

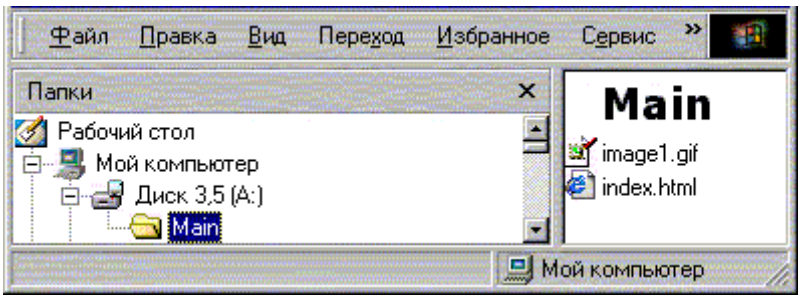


Рис. 2.5

Пример 2. Создать гиперссылку на файл **image1.gif** из файла **index.html**. Расположение файлов показано на рис. 2.6.

В данном случае гиперссылка будет иметь вид:

смотри рисунок

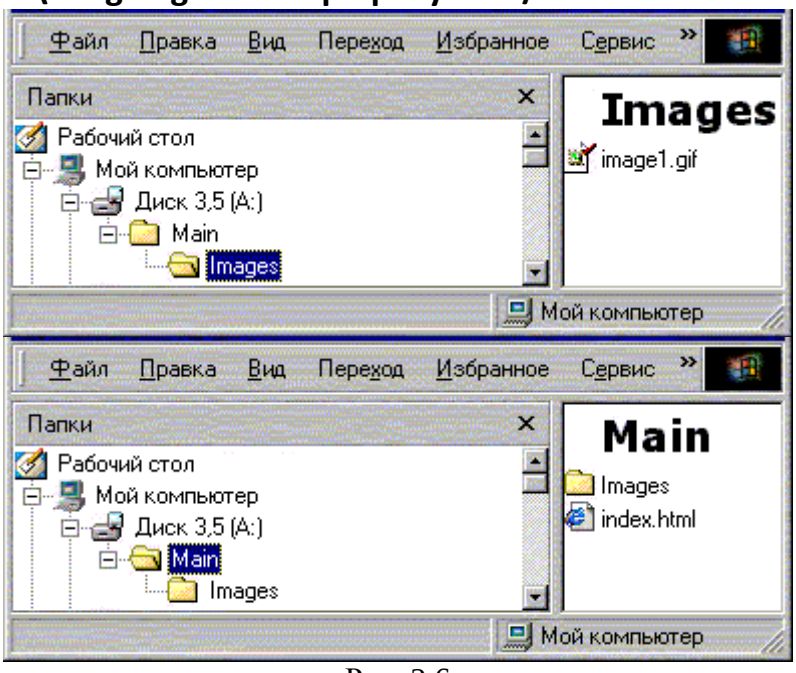


Рис. 2.6

Пример 3. Создать гиперссылку на файл **image1.gif** из файла **index.html.** Расположение файлов показано на рис. 2.7**.**

В данном случае гиперссылка будет иметь вид**:** **смотри рисунок**

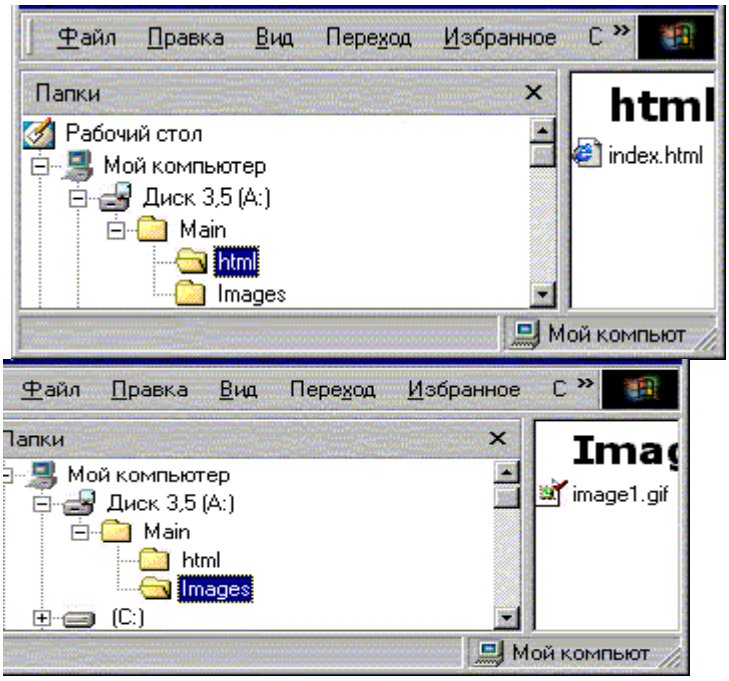


Рис. 2.7

2.3.3 Навигация и дизайн

Навигационные элементы должны удовлетворять принципу единства, т.е. иметь одинаковый размер, расположение и оформление на всех страницах. Текстовые ссылки подчиняются тем же правилам, что и графические. Для удобства использования текстовые ссылки зачастую дублируют графические внизу страницы. Навигационные элементы располагают таким образом, чтобы минимизировать движение мыши пользователя. Кроме того, существует неписанное правило о том, что любая информация должна быть получена при использовании не более чем трех щелчков мыши.

Графические кнопки не должны быть громоздкими и вычурно оформленными. Не следует забывать о том, что навигация – служебный элемент сайта.

Принцип динамики диктует расположение элементов навигации сверху или слева, однако на домашней странице навигационная панель может быть расположена и в центре страницы.

2.3.4 Классификации ссылок

Гиперссылки могут быть классифицированы по различным признакам.

По *целевому* назначению ссылки разделяются на:

- внутри страничные;
- внутри сайта;
- внешние (на внешние ресурсы).

По *структурированности* ссылки разделяются на:

- структурированные (образующие навигационные панели или списки);
- неструктурированные (расположенные в тексте, «оживляющие» страницу и стимулирующие пользователя к тщательному просмотру страницы).

По *динамичности* ссылки разделяются на:

- статические;
- динамические (при выборе пользователем какой-либо ссылки, он получает приглашение воспользоваться дополнительными ссылками, которые тематически связаны с уже выбранной. ("Вам также понравится…" или "Вас может заинтересовать и этот товар…." и т. п.)

2.3.5 Виды ссылок

- Текстовые, образуют текстовую панель или используются для дублирования графических ссылок (обычно внизу страницы).
- Графический текст – текст, созданный графическими средствами. Преимуществом является произвольный дизайн, а недостатком – затраты времени на загрузку и ограничение доступа в неграфических средах.
- Графические кнопки, имеющие несколько состояний.
- Пиктограммы (чисто графические или в сочетании с текстом). Они загружаются быстрее графических кнопок, т.к. их размер 6 – 10кб. Вместо текста можно использовать атрибут title. Рекомендуется использовать общепринятый набор пиктограмм с изображением стрелок, дома (переход на домашнюю страницу), письма (переход к электронной почте) и т. п.
- Карты изображений. Преимуществом является возможность создать горячие области неправильной формы, а недостатком – проблемы в невизуальных средах;
- Рекламные баннеры.
- Выпадающее меню – раскрывающийся список ссылок.
- Меню на основе DHTML – может быть многоуровневым.
- Ролловеры (изменяющееся изображение или текст при наведении и сведении мыши и переход по щелчку).
- Горячие точки, которыми могут стать любые элементы при использовании соответствующей конструкции JavaScript (на примере элемента p):

  ** переход на yahoo **

При этом курсору можно придать вид, характерный для наведения на гиперссылку – руки. Это делается с помощью стилевого описания **cursor:pointer**.

2.3.6 Элементы теории навигации

Навигация – это наука (или искусство) перемещения из одного места в другое. Если географическая навигация имеет ориентиры в виде долготы и широты, отсчитываемых от Гринвича и экватора, то в информационном пространстве такие удобства отсутствуют.

Навигация в Web должна помочь пользователям определить свое текущее местоположение, направление, куда им следует двигаться и то, как вернуться в ранее посещенное место.

Цель навигации – добраться до места назначения с наименьшими затратами (времени на щелчки мыши). Хорошая навигация должна снабжать пользователя полной информацией о его местоположении и возможностях дальнейшего передвижения на каждом шаге взаимодействия с сайтом и страницей.

При посещении страницы пользователь должен иметь ответы на следующие вопросы.

Где я?

Местоположение в Web определяет **URL** (**Uniform Resource Locator**). Чтобы улучшить навигацию нужно использовать простые, понятные и запоминающиеся **URL. URL** может быть прочитан в адресной строке браузера, иногда его помещают в строку статуса web-страницы. К сожалению, этот ответ на вопрос «Где я?» не всегда может быть полезен или понятен (как в случае динамической генерации страниц), т.к. не содержит никакой информации о связях между страницами.

Точную информацию о своем местонахождении кроме **URL** пользователи могут получить из обозначений страницы. Как правило, в верхней части каждой страницы располагаются ясные обозначения сайта, его раздела и страницы.

Общепринятое соглашение в области дизайна предусматривает расположение в верхнем левом углу страницы логотипа с названием организации, зачастую совпадающие с названием сайта. Логотипы обычно располагают на каждой странице, поскольку они выступают в качестве индикаторов сайта. Для логотипа всегда должно быть определено действие, позволяющее пользователю вернуться на домашнюю страницу сайта. Уместно также сделать подсказку в виде атрибута **title**, указывающего на то, что логотип – это ссылка на домашнюю страницу. Для возвращения на домашнюю страницу также следует использовать более явную возможность, которая заключается в размещении на странице кнопки «Домой».

Еще одним способом указания местоположения является применение обозначения раздела и страницы. Выбранный раздел должен иметь в навигационной панели более блеклое обозначение по сравнению с остальными и не должно быть возможности нажать кнопку, обозначающую раздел. Разделы следующих уровней иерархии также должны быть указаны более блеклым обозначением. Заголовок текущей страницы обычно располагается в ее верхней части и каким-то образом выделяется. Расположение заголовков текущих страниц и их оформление должны быть неизменными на каждой странице.

Иногда на странице используют так называемый. глубиномер, который показывает пользователю, на каком уровне иерархии он находится. Например, пользователь находится на странице **Trainer**

Home| Products| Robots| Trainer

при следующей иерархии:

About
Products----…..
Robots----Butler
Trainer
Security
Friend
News
Jobs

Для указания местоположения иногда применяют цветовое кодирование или темы. В первом случае разделы сайта отличаются цветовым оформлением, а во втором – каждый раздел сайта имеет какое-либо обозначение (чаще всего изображение), характерное для данного раздела. Например, в разделе продукции помещается картинка продавца, в зоне информации для инвесторов – изображение брокера, рассматривающего облигацию и т. д.

Недостатком первого метода является излишняя пестрота оформления, а второго – опасность использования метафор, понятных только разработчикам.

Где я был?

Для указания посещенных страниц изменяется цвет ссылки с синего (непосещенная ссылка) на фиолетовый (посещенная). Эти цвета приняты по умолчанию, поэтому менять их нежелательно, хотя иногда дизайн сайта требует применения других цветосочетаний.

Помимо изменения цвета ссылок, web-браузер также отслеживает места, где был пользователь, используя журнал. Описание каждой посещенной страницы сохраняется в журнале браузера и пользователь при

помощи кнопок Назад и Вперед может переходить по записям в списке. При создании аналогичных кнопок в странице, нужно указывать более точное назначение обратных ссылок, например, Назад к Изделиям.

Вероятно, наиболее передовым способом указания пользователю его последнего посещения является использование cookie. Cookie – это небольшая по объему текстовая информация, сохраненная в системе пользователя сервером. Основной целью использования cookie является сохранение информации об установках пользователя, которые затем могут использоваться для создания комфортного окружения и персонализации, а также предоставить некоторую помощь в навигации.

Куда я могу пойти?

Способ предоставления вариантов возможных переходов представляется достаточно важным. Доступные варианты не должны быть скрыты за сложными многоуровневыми меню.

Чаще всего навигационные панели располагают вверху или слева страницы. В первом случае возникает проблема вытеснения элементов навигации при прокрутке страницы. Решениями этой проблемы является наличие ссылок в верху страницы, закрепление элементов навигации с помощью средств CSS или дублирование графической панели текстовыми ссылками внизу страницы.

При размещении навигации слева может не хватить места для полноценного просмотра страницы. В этом случае можно использовать дополнительные окна для открытия в них «широких» страниц. При этом необходимо разместить в этих окнах специальные кнопки «закрыть», позволяющие убрать эти окна с экрана.

Навигация в центре обычно уместна только на домашней странице.

Другая схема навигации предусматривает использование всплывающих окон. Всплывающие окна применяются при баннерной рекламе.

2.4 Формы

Форма предназначена для обмена данными между пользователем и сервером. Область применения форм не ограничена отправкой данных на сервер, с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

Документ может содержать любое количество форм, но одновременно на сервер может быть отправлена только одна форма.

Тег **<form>** устанавливает форму на web-странице.

Для отправки формы на сервер используется кнопка **Submit** или нажатие клавиши **Enter** в пределах формы. Если кнопка **Submit** отсутствует в форме, клавиша **Enter** имитирует ее использование.

Когда форма отправляется на сервер, управление данными передается программе, заданной атрибутом **action** тега <**form**>. Предварительно

браузер подготавливает информацию в виде пары "имя=значение", где имя определяется атрибутом **name** тега<**input**>, а значение введено пользователем или установлено в поле формы по умолчанию. Если для отправки данных используется метод **Get**, то данные формируются в адресной строке. Кроме метода **Get** может использоваться метод **Post.**

Тег **<input>** является одним из разносторонних элементов формы и позволяет создавать разные элементы интерфейса и обеспечить взаимодействие с пользователем.

Тег **<input>** предназначен для создания текстовых полей, различных кнопок, переключателей и флажков. Хотя элемент **<input>** не требуется помещать внутрь контейнера **<form>**, определяющего форму, но если введенные пользователем данные должны быть отправлены на сервер, где их обрабатывает серверная программа, то указывать **<form>** обязательно.

Основной атрибут тега **<input>**, определяющий вид элемента - **type**. Он позволяет задавать следующие элементы формы:

- текстовое поле (**text**), поле для ввода текста (по умолчанию вмещает 20 знаков)
- поле с паролем (**password**),
- переключатель (**radio**), используется, когда следует выбрать один вариант из нескольких предложенных.
- флажок (**checkbox**), позволяет выбрать более одного варианта из нескольких предложенных
- скрытое поле (**hidden**),
- кнопка (**button**),
- кнопка для отправки формы (**submit**),
- кнопка для очистки формы (**reset**),
- поле для отправки файла (**file**)
- кнопка с изображением (**image**)
- поле для задания адреса электронной почты (**e-mail)**
- поле для ввода числовых данных в диапазоне от min до max (**number**)
- поле для ввода числовых данных в диапазоне от min до max индикатор (бегунок) (**range**)
- поле для ввода телефонного номера по шаблону (**tel**)
- поле для ввода даты (**date**)
- поле для ввода времени (**time**)

Атрибуты тега **<input>** представлены далее:

- **autofocus** -устанавливает фокус в поле формы.

- **checked** -предварительно активированный переключатель или флажок.
- **disabled** - блокирует доступ и изменение элемента.
- **action** - определяет адрес обработчика формы.
- **method** - метод протокола HTTP. Сообщает браузеру каким методом следует передавать данные формы на сервер.
- **max** - верхнее значение для ввода числа или даты.
- **maxlength** - максимальное количество символов разрешенных в тексте.
- **min** - нижнее значение для ввода числа или даты.
- **name** - имя поля, предназначено для того, чтобы обработчик формы мог его идентифицировать.
- **pattern** - устанавливает шаблон ввода.
- **placeholder** - выводит подсказывающий текст.
- **readonly** - устанавливает, что поле не может изменяться пользователем.
- **required** - обязательное для заполнения поле.
- **size** - ширина текстового поля.
- **src** - адрес графического файла для поля с изображением.
- **step -** шаг приращения для числовых полей.
- **tabindex** - определяет порядок перехода между элементами с помощью клавиши **tab**.

Код html-файла, позволяющий создать форму, представленную на рис. 2.8, приведен далее.

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta charset="utf-8">
  <title>Ter INPUT</title>
 </head>
 <body>
<form name="test" method="post" action="input1.php">
  <p>Ваше имя:<br>
   <input type="text" size="40">
  </p>
  <p>Каким браузером в основном пользуетесь:<br>
   <input type="radio" name="browser" value="ie"> Internet Explorer<br>
   <input type="radio" name="browser" value="opera"> Opera<br>
   <input type="radio" name="browser" value="firefox"> Firefox<br>
```

```
 </p>
 <p>Комментарий<br>
  <textarea name="comment" cols="40" rows="3"></textarea></p>
 <p><input type="submit" value="Отправить">
  <input type="reset" value="Очистить"></p>
</form>
</body>
</html>
```

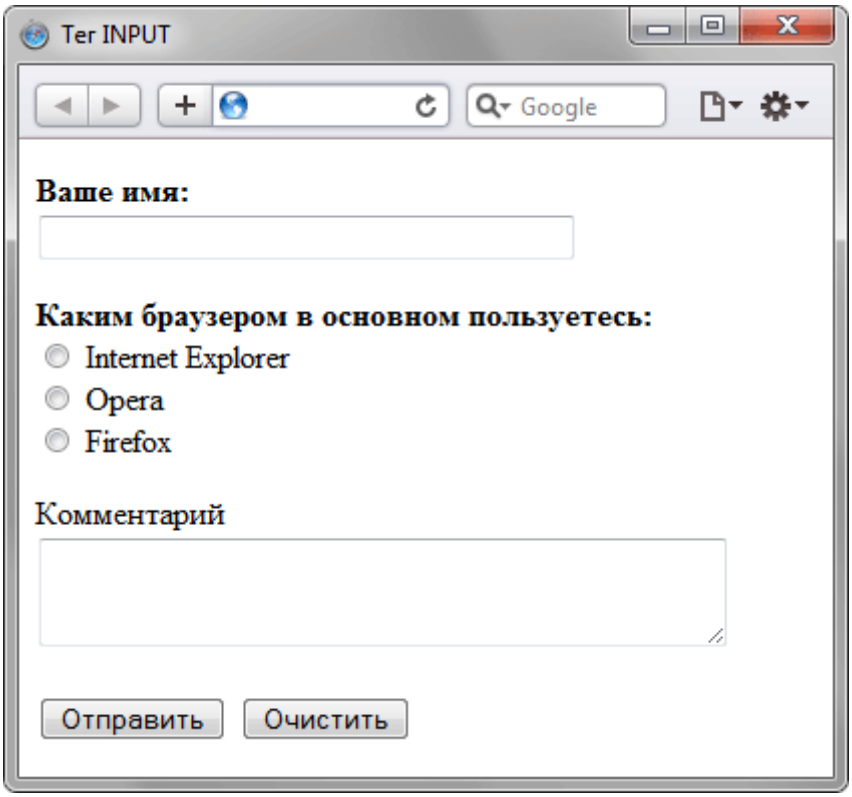


Рис. 2.8

Для создания выпадающих списков используется тег **<select>**, а элементы выпадающего списка определяются с помощью тега **<option>.**

Фрагмент кода html-файла, позволяющий создать форму с выпадающим списком, представленную на рис. 2.9, приведен далее.

```
<p> Выберите ваш пол </p>
<form>
<select name="sex" >
<option value="m"> Мужской </option>
<option value="f"> Женский </option>
</select>
</form>
```

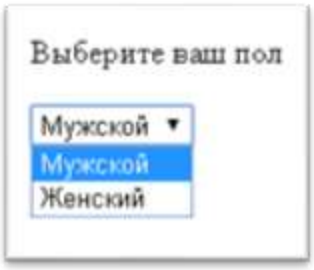


Рис. 2.9

С помощью атрибута **multiple** можно указать, что в выпадающем списке могут быть выбраны одновременно несколько элементов.

Создание секции в форме является распространенной задачей. Для того, чтобы озаглавить группу элементов формы, нужно с помощью тэга **<fieldset>** сгруппировать желаемую часть формы и затем с помощью тэга **<legend>** установить заглавие.

Фрагмент кода html-файла, позволяющий создать форму, содержащую секцию, представленную на рис. 2.10, приведен далее.

```
<form>
<table width=30% align="center">
<tr>

      <td><fieldset>
      <legend>Данные о пользователе</legend>
      Имя: <br />
      <input type="text" name="firstname" /><br />
      Фамилия:<br />
       <input type="text" name="lastname" /><br />
      Отчество: <br />
      <input type="text" name="lastname" /><br />
      <p>Укажите Ваш пол:</p>
      <input type="radio" name="s" value="m" checked /> Мужской<br />
      <input type="radio" name="s" value="f" /> Женский</fieldset></td>
</tr>
</table>
</form>
```

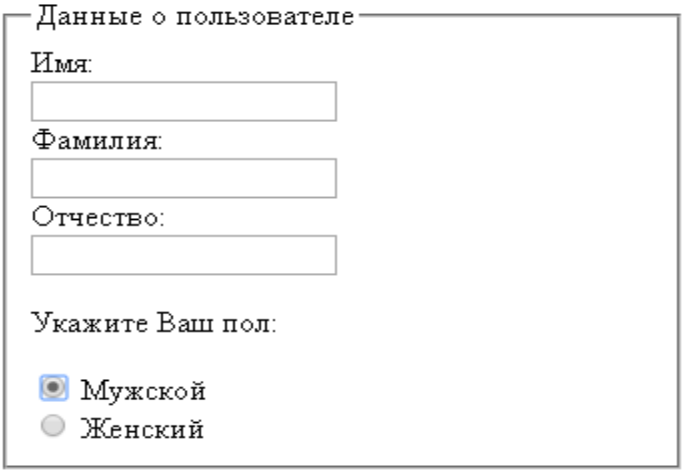

Данные о пользователе
Имя:
Фамилия:
Отчество:
Укажите Ваш пол:
Мужской
Женский

Рис. 2.10

2.5 Мультимедиа в HTML5

Аудио-и видеоматериалы являются важной частью современного Интернета. Подкасты, аудиокомментарии и видеоинструкции широко используются в web-страницах, и до появления HTML5, для их воспроизведения требовались специальные браузерные плагины, которые, в свою очередь, нуждаются в больших объемах шаблонного html-кода.

В HTML5 появились новые механизмы встраивания аудио- и видеофайлов в страницу.

Для расположения аудио и видео на web-страницах используются дескрипторы **<audio>** и **<video>,** использующие для воспроизведения встроенные средства браузера**.**

В коде, приведенном далее определяется элемент **<audio>** и атрибут **control** указывает на необходимость отображения кнопки управления воспроизведением. Далее для элемента определяются несколько файлов-источников (OGG и MP3 версии), а затем используется ссылка для загрузки файла MP3 на случай, если браузер не поддерживает элемент **<audio>**.

```
<audio id=”drums” controls>
  <source src=”sounds/ogg/drums.ogg”  type=”audio/ogg”>
  <source src=”sounds/mp3/drums.mp3”  type=”audio/mpeg”>
  <a href=”sounds/ mp3/drums.mp3”>Download drums.mp3</a>
<audio>
```

Если открыть страницу в HTML5-совместимом браузере, каждый элемент списка будет представлен отдельным аудиопроигрывателем. Браузер обеспечивает воспроизведение аудио при нажатии кнопки **Play.**

Дискриптор **<video>** аналогичен элементу **<audio>.**

В контейнерном элементе **<video>** указываются источники видео.

```
<video controls>
<source src=”video/h264/01_blur.mp4”>
  <source src=”video/theora/01_blur.ogv”>
  <source src=”video/webm/01_blur.webm”>
</video>
```

Установка атрибута **controls** указывает браузеру на необходимость отображения элементов управления видео. Если же атрибут не указан, то будет виден лишь первый кадр (или изображение, указанное атрибутом **poster**) и воспроизведение видео будет возможно или с помощью JavaScript, или собственными средствами управления.

Атрибут **autoplay** обеспечит автоматическое воспроизведение видео браузером. Рекомендуется использовать его с осторожностью, так как это может быть неудобно для пользователя, если у него низкая скорость соединения (например, мобильный телефон).

Атрибут **muted** позволяет выключить звук.

2.6 Графика на web-странице

HTML использует для вывода графических изображений дескриптор **<img>,** который имеет следующие обязательные атрибуты:

src="URL"–источник (содержит адрес файла с рисунком);

alt – альтернативный текст, который выводится при невозможности браузера отобразить рисунок или для пользователей, которые отключают просмотр графики;

width и **height** – ширина и высота изображения. Необходимость их использования объясняется тем, что при наличии этих атрибутов браузер может скомпоновать web-страницу без предварительной загрузки графики.

Для выравнивания изображения, находящегося в *тексте,* применяется атрибут **align** со значениями **top, bottom, middle**, обеспечивающими выравнивание относительно базовой линии строки.

Значения атрибута **left** и **right** дают возможность выравнивания изображения, находящегося *вне* текста (*плавающего* изображения). Эти виды выравнивания иногда применяются в сочетании с дескриптором **
** и его атрибутом **clear**, имеющим значения **left**, **right** и **all**, что позволит прервать текст и выделить плавающее изображение (рис. 2.11).

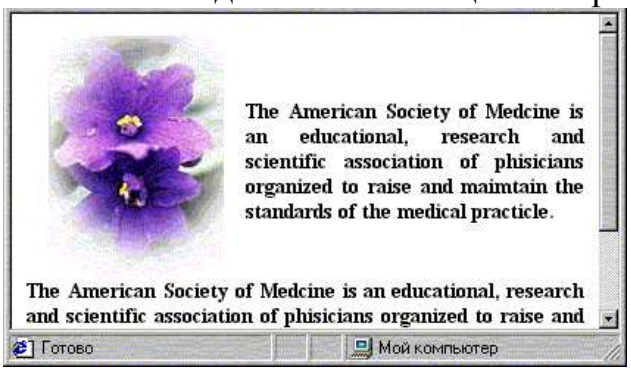


Рис. 2.11

Кроме атрибута **align** Консорциум WWW рекомендует использовать свойство **CSS - float** со значениями **left** и **right**, которое позволяет создать не только плавающее изображение, но и любой другой элемент HTML.

Для установки вертикальных и горизонтальных промежутков между текстом и изображением используются атрибуты **vspace** и **hspace**, соответственно.

Атрибут **border,** имеющий значения в пикселях, позволяет устанавливать или снимать рамку вокруг изображения.

Для размещения изображения в центре страницы можно поместить его в контейнер ** ** с атрибутом **align,** имеющим значение **center.**

2.6.1 Работа с графическими изображениями

Можно назвать следующие источники графических изображений на WEB-странице.

1. Создание графического изображения. Обычно графические изображения создаются с помощью графических редакторов (векторных или растровых), затем векторные изображения экспортируются в растровый формат, а затем и те, и другие оптимизируются с целью достижения компромисса между качеством и объемом файла.

Наиболее распространенными графическими редакторами являются Adobe PhotoShop, Adobe Illustrator, Adobe ImageReady, CorelDraw.

2. Сканирование фотографий или других объектов. Для сканирования лучше использовать цветной сканер с достаточно высоким разрешением. Результат сканирования можно обработать графической программой (изменить размер, цвета, добавить обрамление, эффекты).

3. Использование хранилищ изображений. Хранилища изображений – это картинки из коллекций (clip art) и фотографии, доступные за определенную плату или бесплатно и готовые к использованию.

4. Использование гиперссылки на изображение в web или самого изображения. В первом случае существует нежелательная возможность изменения адреса ссылки, а во втором нужно помнить об авторских правах создателей страниц и заручаться их разрешением.

5. Использование цифровой камеры или фотоаппарата.

Размер графических изображений не должен превышать 20-30 Кб, а пиктограмм (графических миниатюр)–2-6 Кб, чтобы общий размер файла не превышал 150 Кб.

2.6.2 Форматы графических файлов

GIF (Graphic Interchange Format) – первый тип графических файлов, поддерживаемых WEB. В GIF используются алгоритмы сжатия без потерь информации. Степень сжатия регулируется несколькими параметрами: глубиной цвета, сглаживанием и размыванием границ. Увеличение глубины цвета, наличие сглаживания и размывания границ приводят к увеличению объема файла. GIF может иметь глубину цвета от 1 до 8 бит, и таким образом отображать от 2 до 256 цветов.

GIF целесообразно применять для изображений с небольшим количеством цветов или для штриховых изображений.

GIF поддерживает чередование, при котором графическое изображение будет появляться на экране не сразу, а сначала неясно, а затем более отчетливо.

GIF позволяет использовать прозрачность.

Еще одной особенностью этого формата является возможность хранить в одном файле несколько изображений, которые используются для

создания анимации. Анимированный GIF вставляется в web-страницу как обычный графический объект с помощью дескриптора **<img>.**

JPEG (Joint Photographic Experts Group) был создан группой экспертов по фотографии для высококачественного сжатия. Он позволяет хранить изображения с глубиной цвета 24 бита, т.е. отображать порядка 16 млн. цветов. JPEG использует сжатие с потерями и имеет характеристику, называемую уровнем качества. Она находится в диапазоне от 0 (максимальное сжатие) до 100 (максимальное качество). Большинство JPEG файлов сохраняются с уровнем качества от 50 до 100.

JPEG целесообразно применять для фотографических изображений или для иллюстраций, содержащих плавные переходы цветов.

JPEG поддерживает прогрессирующую передачу, которая применяется с теми же целями, что и чередование в GIF.

PNG (Portable Network Graphics) – переносимая сетевая графика. Этот формат одобрен консорциумом W3C. Таким образом, ведущие браузеры в скором будущем поддержат этот формат. Формат PNG является одним из самых гибких графических форматов. PNG поддерживает чередование и прозрачность.

В этом формате используется алгоритм сжатия без потерь информации. В PNG-изображениях может использоваться разная глубина цветопередачи: от 8 до 32 бит.

2.6.3 Применение графических изображений [4]

2.6.3.1 Заголовки

Строка в заголовке окна браузера, заглавие – это первый заголовочный элемент, который согласно спецификации HTML обязательно должен быть отображен в тексте документа с помощью дескриптора **<title>**. Заглавие не относится к дизайну страницы, но необходимо для поисковых систем и автоматических сборщиков информации о сайтах.

Заголовки на web-страницах могут быть помещены с помощью иерархии дескрипторов **<h1>...<h6>**. Однако очень редко страницы используют такую глубокую иерархию заголовков, а параметры оформления этих заголовков часто не отвечают художественным требованиям сайта. Кроме того, стандартные заголовки HTML не могут быть собраны в оглавление или пронумерованы. Такие возможности доступны для XML-документов со стилевыми спецификациями на XSL.

В связи с этим наиболее распространенным вариантом является использование графической вставки в роли заголовка. Заголовок с его оформлением должен быть интегрирован в композицию страницы и составлять с ней единое целое, так, какие-либо элементы заголовка могут повторяться в элементах страницы или в заголовках нижних уровней. В

качестве примера использования заголовков может послужить заголовок, приведенный на рис. 2.12.

Рис. 2.12

2.6.3.2 Элементы навигации

Простейшей разновидностью навигационной панели для сайта с древовидной топологией является список текстовых ссылок, расположенный на первой странице и иногда сопровождающийся краткой аннотацией по каждой ссылке.

Для дублирования на остальных страницах можно применить более компактное средство – навигационную панель с кнопками (или другими графическими объектами, выполняющими роль кнопок). Количество кнопок в таких панелях не должно быть велико.

Графические средства оформления кнопок отделяют их друг от друга с помощью линий, рамок и т. д., либо располагают рядом с активной кнопкой треугольник, кружок или какое-то другое символическое изображение, при этом важным моментом является однородность этих элементов: одинаковая длина или высота кнопок и сопровождающих их элементов.

Кнопки могут быть оформлены с использованием небольших стилизованных рисунков, как в случае сайта www.yahoo.com.

Иногда, для сложной системы навигации используются либо элементы HTML форм, либо динамический HTML с применением программирования на JavaScript (ролловеры или выпадающие меню).

Для сайтов с линейной, а не древовидной навигацией необходима пара ссылок, ведущих на предыдущую и последующую страницы (рис. 2.13).

Рис. 2.13

Однако чаще, линейная навигационная панель включает в себя строку ссылок на все страницы, указывая на место нахождения пользователя в этой цепочке, тем самым поддерживая его ориентацию. Примером может служить организация ссылок на страницы в поисковых системах или оригинальная система навигации и ориентации, приведенная на рис. 2.14.

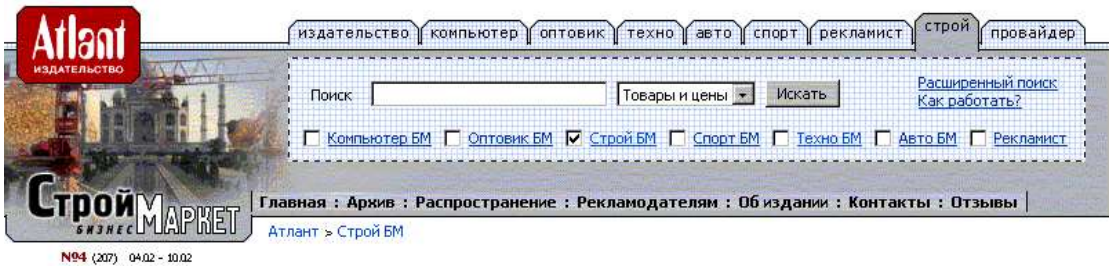


Рис. 2.14

2.6.3.3 Фон

При использовании фона нужно руководствоваться следующими принципами.

Фон не должен "заглушать" передний план не только в смысле подбора цветов, но и своей регулярностью, которая будет неизбежно присутствовать при использовании текстурных элементов с громоздким или навязчивым узором.

Распространенными типами фоновых изображений являются фоны, размножаемые только по одному измерению. При размножении фона только по вертикали получается полоса фона в левой части экрана, (так называемые "краевые обои"). Их можно использовать для навигационной панели, логотипа или баннера (рис. 2.15).

Кроме фонов, размножаемых только по одному измерению, существуют неповторяющиеся фоны, которые занимают весь экран. Обычно это бледные изображения, которые не мешают чтению текста. Фоновые изображения не должны превышать 20-30 Кб, чего можно достичь сжатием в JPEG с потерей качества, либо в GIF с уменьшением глубины цветности.

Рис. 2.15

2.6.3.4. Логотип

Логотип – это графический и (или) шрифтовой знак, который принадлежит фирмам, сайтам, организациям, товарам, услугам, а также отдельным людям. Важность этого элемента объясняется тем, что большинство страниц и сайтов современного Интернета не нуждаются в

обильной рекламной графике или не могут себе позволить занимать информационную площадь страниц графическими изображениями. Поэтому, логотип зачастую является единственным графическим элементом на странице.

К логотипу предъявляется два основных требования – это узнаваемость (легкость узнавания и последующего запоминания) и оригинальность. Первое требование может быть достигнуто при наличии внутренней логики и простоты (компактности) композиции, а оригинальность – необычностью композиции.

Логотип должен быть иллюстративным, т.е. отражать в какой-либо степени тот объект, которому он принадлежит. Отправной точкой может служить первая буква или аббревиатура названия фирмы в качестве основы графической части знака.

Композиционно логотип обычно состоит из графического знака и подписи, обычно, названии компании. Текст располагается справа или под изображением.

Однако существуют исключения, такие, например, как логотип формы Microsoft, состоящий только из названия фирмы и небольшим графическим вырезом в одной из букв.

На рис. 2.16 приведен логотип сайта города Exeter в Великобритании.

Рис. 2.16

2.6.3.5. Баннер

Баннер (рис. 2.17) представляет собой рекламную графическую вставку и способствует признанию рекламируемого продукта и увеличению продаж через Интернет. Реклама зачастую является способом окупить затраты на создание и поддержку сайта, особенно это относится к контент-сайтам. Само слово баннер, означающее "заголовок", действительно является заголовком для тех страниц, на которые он приглашает перейти.

Рекламный баннер обычно имеет размер 468×60 пикселов и не должен превышать 8-10 Кб. В баннерах часто используется GIF анимация и насыщенная цветовая гамма. Обычно баннер располагают в самом верху страницы. Небольшие баннеры размером 125×125 пикселов располагают в правой колонке страницы.

Рис. 2.17

Главным мерилом эффективности баннера является отношение числа посетителей, щелкнувших по нему, к общему числу посетивших страницу с

баннером. Эта величина, обозначаемая по-английски CTR (click through ratio), может быть измерена по статистике сервера.

2.6.3.6. Визуал

Визуал – это графический элемент, выступающий в роли тематической иллюстрации, фотографики, могущей служить эмблемой, сопровождением к тексту или просто фоновым элементом.

С другой стороны, в отличие от иллюстраций в научных и деловых документах, однозначно соотносящихся со смыслом текста, темы визуалов могут быть практически любыми.

Из эстетических требований следует отметить необходимость достаточного текстурного контраста между изображением и его окружением. Самым популярным источником для визуалов служат фотографии, часто сопровождаемые эффектами прозрачности или размытыми краями. Кроме фотографики источником для визуалов является и рисованная графика.

2.6.3.7. Миниатюры

К миниатюрам можно отнести уменьшенные копии полноразмерной графики, кнопки, маркеры, различные пиктограммы: новинок, реконструкции, перемещения. Эти графические изображения чаще всего создаются в формате GIF и имеют размер 2-6 Кб.

2.6.3.8. Графика иллюстративная и сама по себе

К этим категориям можно отнести сканированные фотографии, репродукции картин, факсимиле рукописей и т.д.

Иллюстративная графика служит для подчеркивания или сопровождения текста. Чтобы она отличалась от обычной, к ней можно добавить тени, рамки и другие эффекты, которые должны быть в строгом соответствии с дизайном страницы.

Второй вид графики - это графические изображения, которые вставляются на страницу не со служебной или декоративной целью, а ради них самих.

Зачастую, для уменьшения времени передачи по сети, на странице располагают миниатюры (thumbnails) – уменьшенные копии хранящихся на странице изображений. Каждая из таких миниатюр связана с ее полноразмерной копией или с web-страницей с полноразмерной картинкой.

Лекции 4,5

Глава 3. Технологии создания web-страниц. Каскадные таблицы стилей (CSS)

Технология каскадных таблиц стилей (Cascading Style Sheets, CSS) была разработана в качестве дополнения к HTML с целью определения внешнего вида документов и сохранения за HTML только функции структурной разметки документов. Система CSS независима от HTML, имеет иной синтаксис и позволяет задавать параметры графического (так же как и текстового, и звукового) представления дескрипторов HTML.

Таблицы стилей – это набор элементов оформления, которые применяются к различным частям документа и описывают способ их представления на экране. В принципе, таблицы стилей реализованы во всех функционально-развитых текстовых процессорах.

С помощью таблиц стилей можно форматировать текст, используя методы, приближенные к методам форматирования обычных печатных страниц. Созданные страницы будут корректно функционировать, учитывая некоторые ограничения, связанные с платформами, браузерами, различными размерами экранов и разрешениями мониторов. Однако процесс разработки web-стандартов в скором времени обещает решить эти проблемы.

Принятие в 1996 году Консорциумом W3C CSS первого уровня в качестве стандарта позволило отделить содержание web-страницы (текст, графические изображения и т. д.) от ее оформления (макет страницы и характеристики текста, например, шрифты, цветовое оформление и т. д.). После этого язык HTML вновь стал функционально-ориентированным, а не ориентированным на форму. Стандарт CSS2, принятый в 1998 году, основан на CSS первого уровня и позволяет разработчикам осуществлять контроль над web -страницами на более высоком уровне. Он включает некоторые новые функции, в частности, возможность точно располагать элементы и объекты web -страницы, применять загружаемые шрифты или использовать звуковые таблицы стилей. Применение стилей позволяет также решить многие проблемы поддержки браузеров, т.к. основные компании-разработчики браузеров встроили таблицы CSS в свои программные продукты.

CSS3 - это новый стандарт оформления HTML документов, значительно расширяющий возможности предыдущего стандарта CSS2.1.

Многие возможности, которые были труднодоступны в CSS2.1, то есть требовали использования дополнительных внешних программ (*таких как Adobe Photoshop*), скриптов (*таких как JavaScript*) или специальных ухищрений могут легко достигаться в CSS3 за счет использования новых свойств оформления.

Существует три способа применения таблиц стилей в документе HTML.

- *Встраивание* (*Inline*) , при котором дескрипторы web-страницы дополняются короткими объявлениями формата. Встраивание предоставляет контроль над фрагментом, к которому оно применяется. Атрибут **style** присоединяется к дескриптору HTML, и соответствующий фрагмент страницы будет выводиться браузером с применением формата, указанного стилем.
- *Внедрение* (*Embed*) обеспечивает контроль над страницей HTML. Использование дескриптора **<style>** в пределах раздела **<head>** позволяет описать атрибуты стиля, применяемого ко всей странице.
- *Связывание* (*Link*), известное также как применение внешнего листа стилей. В этом случае связанный документ использует эталонную таблицу стилей, применяемую для всего сайта и хранящуюся в файле с расширением *.css*.

Термин "каскадные" (таблицы стилей) описывает в первую очередь тот факт, что браузер следует определенному порядку (каскаду) при интерпретации стилей. Можно использовать все три типа стилей, и браузер интерпретирует их в следующем порядке: связанный – внедренный – встроенный.

Второй аспект комбинированного применения стилей – наследование. Например, при объявлении в элементе абзаца некоторого цвета шрифта все элементы, расположенные в его пределах, унаследуют этот цвет.

3.1. Встраивание стиля

Описание стиля можно встроить в различные элементы HTML, для которых стиль имеет смысл, например, в дескрипторы абзацев, заголовков, гиперссылок или ячеек таблицы.

При встраивании в элемент добавляется атрибут **style**, значением которого является свойство и его значение, разделенные двоеточием. Таких пар – "*свойство: значение*" может быть несколько, в этом случае они разделяются точкой с запятой. Значение атрибута **style** заключается в кавычки.

Два элемента **div** (раздел) и **span** (интервал) являются универсальными элементами-контейнерами и позволяют применять стили к фрагментам текста, как показано в примерах 1 - 3.

Пример 1

```
<p style="font-size: 2em; color: #ff0000">Красный текст с размером шрифта 2em </p>
<p style="font-size: 1em; background-color: #ffff00">Текст с размером шрифта 1em и желтым фоном</p>
```

Пример 2

** **Текст с размером шрифта x-small ****Текст с размером шрифта large** **

** **Текст с размером шрифта 12pt и малыми прописными буквами** **

** **Текст с размером шрифта 16pt и серифным шрифтом** **

Пример 3

** **Текст с сокращенной формой записи свойств шрифта** **

На рис. 3.1 показано отображение в браузере страницы с текстами примеров 1-3.

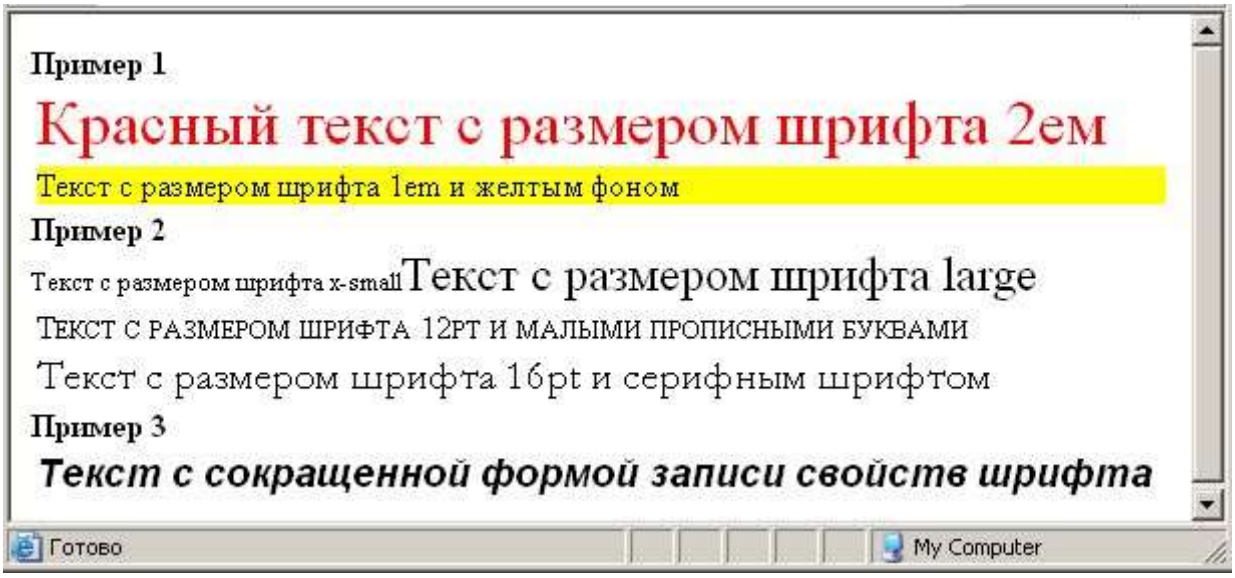


Рис. 3.1

3.2. Внедрение стиля

Для внедрения стиля используется элемент (контейнер) **<style>** в пределах раздела **<head>.**

В приведенном далее примере использованы *правила*, по которым должны быть представлены элементы документа. Каждое правило имеет две основные части: *селектор* и *блок объявлений*. Блок объявлений заключается в фигурные скобки и может содержать одно и более объявлений. Каждое объявление представляет собой сочетание *свойства* и *значения*, которые разделяются двоеточием. Внутри блока объявлений может быть задано несколько объявлений, они отделяются друг от друга точкой с запятой.

Пример 4

<html><head>

```
<title>Пример 4</title>
<style  type="text/css">
body  {
 background:#000000;color:#ffffff
}
h1  {
 font:14pt Verdana; color:yellow
}
</style>
</head>
<body>
<strong>Пример 4</strong><br><br>
Внедренный стиль для раздела body обеспечивает вывод белых символов на черном фоне<br>
<h1>Для вывода заголовка используется внедренный стиль:шрифт Verdana желтого цвета и размером 14 пунктов </h1>
</body></html>
```

В контейнере **<style>** данного примера находятся описания внедренных стилей. Количество стилей может быть произвольным. Атрибут **type** указывает на используемый тип таблиц стилей - CSS.

На рис.3.2 показано отображение текста примера 4 в браузере.

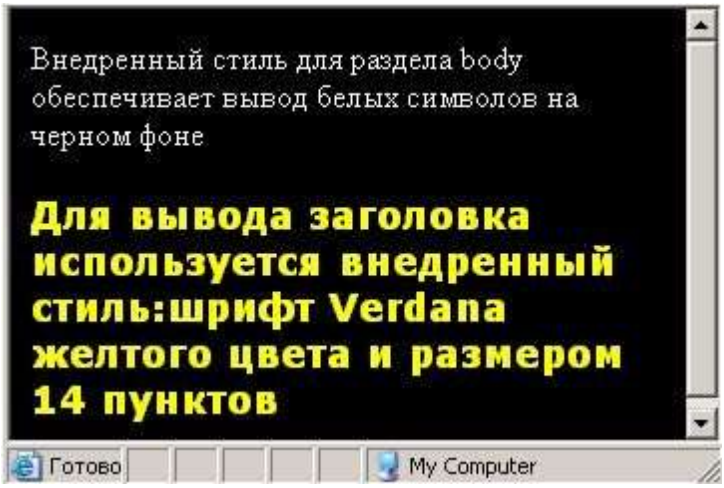


Рис. 3.2

3.2.1. Использование различных селекторов для стилевого оформления

3.2.1.1. Селекторы элементов

Селекторы **body** и **h1** примера 4 являются *селекторами элементов* и определяют область применения данного стиля. Селектор **body** указывает на то, что этот стиль будет применен к разделу **body,** а селектор **h1** на то, что стиль будет применен к тексту внутри всех элементов **h1** документа.

Таким образом, можно объявить разные стили для заголовков и для остального текста.

В примере 4 парами свойство-значение для селектора **body** являются **background:#000000** и **color:#ffffff**, а для селектора **h1 – font:14pt Verdana** и **color:yellow.**

В правиле допускается группирование нескольких селекторов при одном блоке объявлений стиля или же группирование свойств и значений в объявлении стиля.

В примере 5 для того, чтобы придать одинаковые свойства заголовку **h5** и абзацу **p**, используется группирование селекторов.

Пример 5

```
<style type="text/css">
h5, p  {
font-family:Garamond;
font-size:14pt;
color:#660066;}
</style>
```

В примере 6 показано группирование свойств и значений, относящихся к шрифтовому оформлению.

Пример 6

```
<style type="text/css">
body  {
font: italic bold 15pt/18pt Verdana,sans-serif;}
</style>
```

При таком группировании должен поддерживаться определенный порядок следования значений свойств шрифта: наклон (начертание) шрифта, насыщенность шрифта, размер символов, косая черта, межстрочный интервал, семейство шрифтов.

3.2.1.2. Селекторы классов

При необходимости применения правила стилей независимо от элементов используются *селекторы классов*. Классы позволяют также создать несколько наборов стилей на базе одного элемента HTML. Название класса предваряется точкой. В примере 7 класс, позволяющий при необходимости применить желтый фон к ячейке таблицы (элемент **td**) имеет селектор вида: **td.yellow**. Класс **black** того же примера не привязан к конкретному элементу и может быть применен к любому элементу текста. В данном примере класс **black** применен для создания фонов ячеек таблицы в шахматном порядке.

Для задания ширины и высоты таблицы применен селектор элемента **table**.

Для применения класса используется атрибут **class.** Применение классов показано в тексте примера 7. Все ячейки, которым присвоен класс стиля **yellow**, будут иметь желтый фон. Аналогично, ячейки, которым присвоен класс **black**, будут иметь черный фон.

Класс **black** применен также и к элементу **p**, текст которого разместится на черном фоне.

Пример 7

```
<html><head><title>Пример 7</title>
<style>
td.yellow {background-color:yellow;}
.black {background-color:black;}
table {width:45%;height:30%;}
</style>
</head>

<body>
<strong>Пример 7</strong><br>
<table border=2px>
<tr>
  <td class="black"> </td>
  <td class="yellow"> </td>
  <td class="black"> </td>
</tr>
<tr>
  <td class="yellow"> </td>
  <td class="black"> </td>
  <td class="yellow"> </td>
</tr>
<tr>
  <td class="black"> </td>
  <td class="yellow"> </td>
  <td class="black"> </td>
</tr>
</table>
<p class="black" style="color:white;text-align:center">К абзацу применен класс black и встроенный стиль выравнивания содержимого по центру с белым цветом символов</p>
</body></html>
```

На рис.3.3 показано отображение текста примера 7 в браузере.

Рис.3.3

3.2.1.3. Селекторы идентификаторов

Селекторы классов можно применять к любому количеству элементов. В правиле стиля можно использовать селектор идентификатора (селектор ID). В отличие от селектора класса, селектор ID должен быть уникален для данного документа. Селектор ID начинается с символа #. Вместо атрибута **class** в соответствующих элементах HTML используется атрибут **id** Его применение иллюстрировано в примере 8. Селектор **#pict** используется для позиционирования графического изображения, которое помещено в контейнер ** .**

Пример 8

```
<html><head><title>Пример 8</title>
<style type="text/css">
#pict {
position:absolute;
top:100;
left:50;}
</style>
</head>

<body>
<strong>Пример 8</strong>
<div id="pict">
<img src="cats.jpg" alt="cats" width=100 height=100>
</div></body></html>
```

На рис.3.4 показано отображение текста примера 8 в браузере.

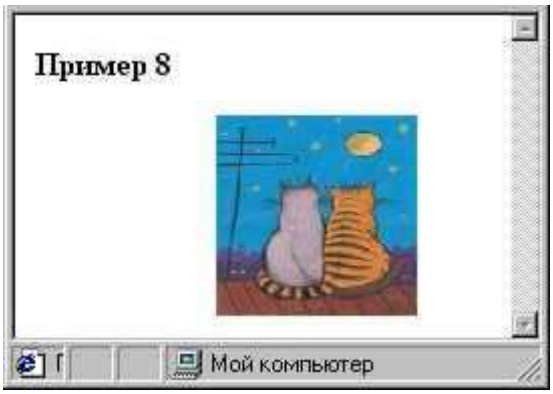


Рис. 3.4

3.2.1.4. Селекторы потомков

Селекторы потомков записываются через пробел после родительского селектора, например, **h1 em {color:red;}**. Это правило сделает красным контент элемента **em**, который является потомком элемента **h1.** В селекторе потомков может быть использовано несколько элементов, например, **ul li a {color:red;}**. Правило применит красный цвет к ссылкам, находящимся в элементах **li** маркированного списка. В примере 19_1 приведены правила с использованием селекторов потомков.

3.2.1.5. Селекторы атрибутов

Селекторы атрибутов могут применяться для выбора элементов на основании их атрибутов и значений этих атрибутов. Селекторы атрибутов были введены в стандарте CSS2 и уточнялись в стандартах CSS2.1 и CSS3.

Для выбора всех заголовков **h1**, имеющих атрибут **class** и окрашивания текста этих заголовков в серый цвет можно использовать правило **h1[class] {color:gray}** или для выделения полужирным шрифтом текст любой гиперссылки, которая имеет и атрибут **title** и атрибут **href** можно записать **a[title][href] {font-weight: bold}**.

Выбор на основании конкретного атрибута производится следующим образом: **a[title="Goto GUT"] {font-weight: bold}**. Полужирным выделением будут отмкчены гиперссылки, значением атрибута **title** которых будет **Goto GUT**.

Возможен также выбор по частичному значению атрибута, например, **img[title~="Рисунок"]{border:1px solid gray}**. Это правило обеспечит нарисование рамки вокруг изображений, в атрибуте **title** которых содержится слово "Рисунок".

3.2.1.6. Псевдоклассы и псевдоэлементы

В CSS2.1 определены псевдоклассы и псевдоэлементы. Традиционно псевдоклассы используются для оформления гиперссылок. Они описывают

свойства ссылок непосещенных (**:link**), посещенных **(:visited**), над которыми находится курсор мыши (т.е. свойства ссылок "при наведении" курсора мыши) **(:hover**) и активных ссылок **(:active**). В примере 9 приведен код HTML, применяющий псевдоклассы для элемента **<a>**. В данном примере непосещенные ссылки имеют черный цвет, активные ссылки окрашиваются в синий, посещенные - в зеленый, а при наведении мыши на ссылку цвет ее меняется на красный и размер шрифта увеличивается до 16 пунктов (рис. 3.5). Порядок расположения псевдоклассов в селекторе важен, так как при его нарушении принцип каскадности применения стилей может привести к игнорированию свойств отдельных псевдоклассов.

Пример 9

```
<html><head><title>Пример 9</title>
<style type="text/css">
a:link  {color:black;}
a:visited {color:#00ff00;}
a:hover  {color:#ff0000;font-size:16pt;}
a:active  {color:#0000ff;}
</style>
</head>

<body>
<strong>Пример 9</strong><br>
<a href="классы_1.html" >Переход к файлу "классы_1.html"</a><br>
<a href="классы.html" >Переход к файлу "классы.html"</a>
</body></html>
```

На рис.3.5 показано отображение в браузере результата действия псевдокласса :hover - наведения мыши на вторую гиперссылку.

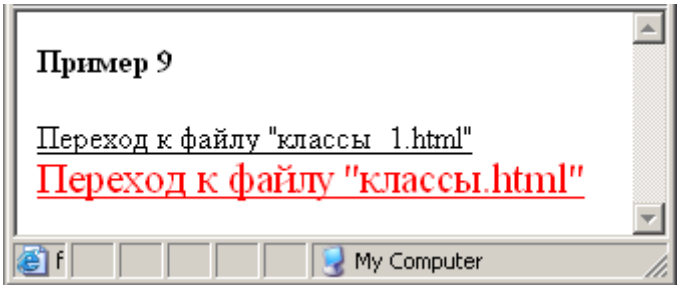


Рис.3.5

К псевдоклассам относится также **:focus**. Этот псевдокласс относится к любому элементу, которому в настоящий момент принадлежит фокус ввода, т.е. который готов принимать ввод с клавиатуры или быть активированным другим способом.

Псевдоэлементы вводят фиктивные элементы в документ, чтобы достигнуть определенных эффектов. В CSS2.1 определены четыре псевдоэлемента, для которых могут быть применены специальные стили: первая буква (**:first-letter**), первая строка (**:first-line**), до (**:before**) и после (**:after**) элемента.

Например, правило **h3:first-letter {font-size:150%}** увеличит первую букву заголовка в полтора раза по отношению к остальному тексту. А правило **p:first-line {color:red}** сделает символы первой строки абзаца красными.

Примеры 10 и 11 иллюстрируют использование псевдоэлементов **:first-letter** и **:first-line.**

Пример 10

```
<html><head>      <title>Пример 10</title>
<style type="text/css">
p:first-letter{
font-size:3em;
font-weight:bold;
color:#ff0000;}
p {line-height:1em;text-align:justify;}
</style>
</head>

<body>
<strong>Пример 10</strong>
<p>Псевдоэлемент first-letter используется для создания буквицы в абзаце. Абзац имеет буквицу  размера 3ем и начертания bold.</p>
<p style="text-indent:200px">Абзац имеет буквицу размера 3 ем, начертания bold  и отступ первой строки 200px. Абзац имеет буквицу размера 3 ем, начертания bold  и отступ первой строки 200px.</p>
</body></html>
```

На рисунке 3.6 показано отображение примера 10 в браузере.

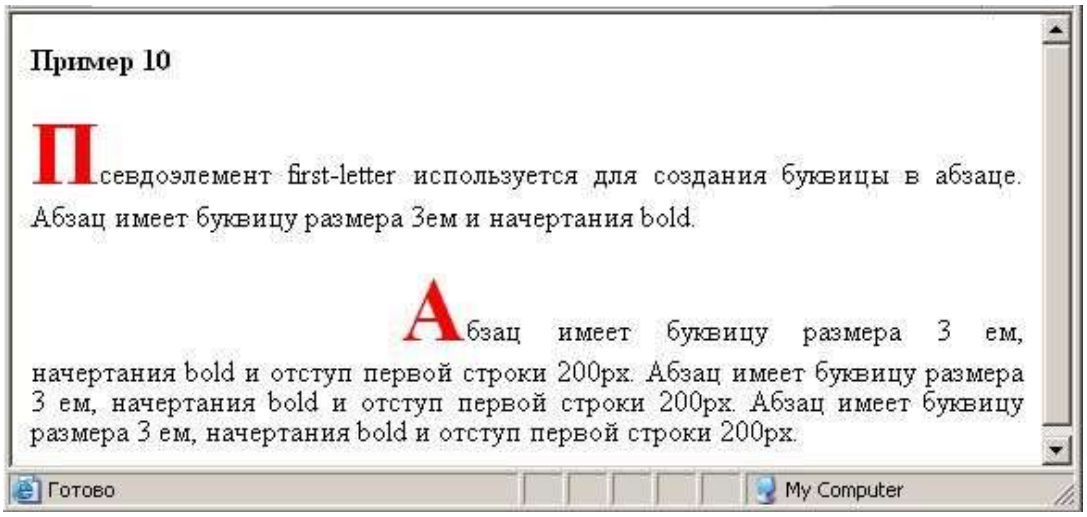


Рис. 3.6

Пример 11

```
<html><head>      <title>Пример 11</title>
<style type="text/css">
p:first-line{
font-weight:bold;}
p {line-height:1em;text-align:justify}
</style>
</head>

<body>
<strong>Пример 11</strong>
<p>Первые строки абзацев стилизованы с помощью псевдоэлемента p:first-line. Первые строки абзацев стилизованы с помощью псевдокласса p:first-line.</p>
<p > Первые строки абзацев стилизованы с помощью псевдоэлемента p:first-line. Первые строки абзацев стилизованы с помощью псевдокласса p:first-line. </p>
</body></html>
```

На рисунке 3.7 показано отображение примера 11 в браузере.

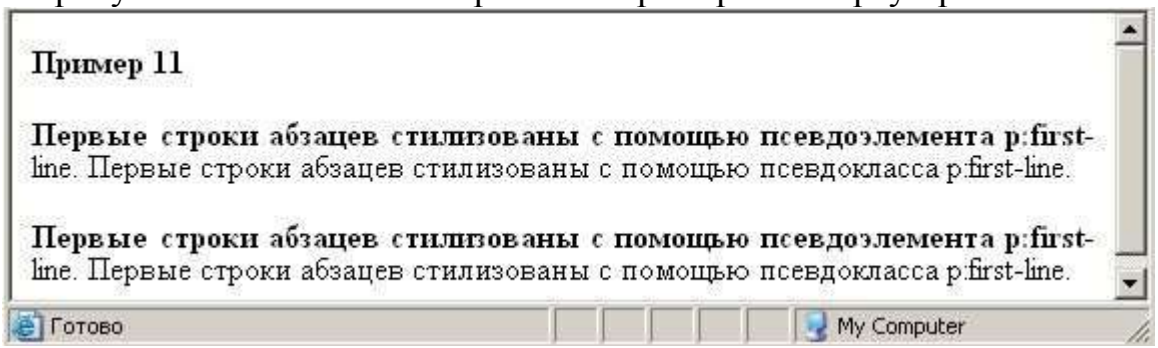


Рис. 3.7

Псевдоэлементы **:before** и **:after** позволяют вставлять *генерируемое содержимое* и затем применять к нему специальные стили. Генерируемое содержимое вставляется с помощью свойства *content*, например, **h2:before {content: "ГУТ"}** добавляет перед заголовками второго уровня текст "ГУТ" или для **body:after {content: "The end"}** добавляет в конец документа текст "**The end**".

3.3. Связывание стиля

При связывании стиля все стилевые описания производятся по тем же правилам, что и при внедрении, но хранятся они в отдельном файле, имеющем расширение *.css*. Сам файл должен находиться в корневом каталоге сайта, в противном случае нужно корректно указать связь с ним.

Преимуществом связывания является то, что стилевые описания применяются ко всем страницам сайта и при необходимости изменения оформления достаточно внести новшества в файл, содержащий эти описания, вместо того, чтобы исправлять все страницы сайта.

В приведенном ниже примере иллюстрируется использование внешней таблицы стилей.

Пример

Файл с именем **my_style.css** содержит *только* правила стилей:

```
body  {
background: #000000;
color: #ffffff;
}
a  {
color: #ff0000; text-decoration:none;
}
```

Документ HTML, который ссылается на этот файл, должен содержать в части **head** следующую ссылку, использующую элемент **link**:

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

Аналогом элемента **link** является директива **@import url (url)**. Директива **@import** также указывает браузеру на необходимость загрузки внешней таблицы стилей. Размещается директива в контейнере **<style>** перед остальными правилами CSS. Примеры использования директив **@import** приведены далее:

```
<style>
@import url ("sheets22.css");
@import url ("http://example.org/library/layout.css");
@import url ("printer.css") print;
</style>
```

Из примера видно, что может быть использовано несколько директив **@import** для присоединения нескольких файлов с таблицами стилей. В последней директиве указано устройство, для которого создана присоединенная таблица.

3.4. Текстовые свойства CSS

1. **text-align -** выравнивание текста со значениями: **left** (по левому краю), **right** (по правому краю), **center** (по центру), **justify** (по обоим краям).

2. **text-indent -** отступ в первой строке блока (абзаца, раздела) со стандартными значениями длины (**pt, px, cm, mm**).

3**. text-decoration -** оформление текста подчеркиванием со значениями: **none** (отсутствует - по умолчанию), **underline** (подчеркивание), **overline** (линия над текстом), **line-through** (перечеркивание), **blink** (мерцание).

4**. text-transform -** перевод букв в верхний или нижний регистр со значениями: **none** (отсутствует - по умолчанию), **capitalize** (первая буква каждого слова становится прописной), **uppercase (**переводит все буквы в верхний регистр), **lowercase** (переводит все буквы в нижний регистр).

5**. text-shadow** - установка эффекта затенения текста со значениями: **none** (отсутствует - по умолчанию), **color left top radius** (цвет затенения с расстояниями слева (справа), вниз (вверх) от текста и радиусом нерезкости).

6**. letter-spacing** - расстояние между символами текста со стандартными значениями длины (**pt, px, cm, mm, em,ex**).

7**. word-spacing** - расстояние между словами со стандартными значениями длины **(pt, px, cm, mm**).

Кроме стандартных единиц измерения длины **px, pt** (равный 0,35мм), **mm, cm**, может использоваться **em** (**1em** равен ширине буквы **m** в шрифте заданного размера), **ex** (**1ex** равен высоте шрифта), **px** (**1px** равен ширине пиксела).

В CSS3 добавлены следующие свойства.

8**.** Добавление тени к тексту с помощью свойства **text-shadow.** Например**, text-shadow:1px 1px 10px #FFAE00**; в качестве значений указывается размер тени по горизонтали и вертикали, радиус размытия (не обязательно) и цвет тени.

9. Свойство **word-wrap:break-word**; позволяет указать, что длинные слова, выходящие за границы элемента, должны разделяться и переноситься на новую строку.

10. Разбиение текста на столбцы происходит с помощью свойств **column-count**, расстояния между столбцами - **column-gap,** ширины столбцов - **column-width** (в обычных единицах длины, чаще в пикселях), оформление столбцов происходит с помощью свойства **column-rule,**

которое задает ширину, стиль и цвет границы между столбцами (**column-rule:5px dotted #ffff00).**

(*Данные свойства поддерживаются в браузерах IE10+ и Opera. Для браузеров Chrome и Safari перед свойством требуется добавить префикс -webkit, а для Firefox префикс –moz)*

Например,

```
{
column-count:3;
-moz-column-count:3;
-webkit-column-count:3;}
```

Пример 12

```
<html><head><title>Пример 12</title>
<style type="text/css">
p.class1 {
text-align: justify;
text-indent: 20pt;
color: #c0c0c0;
background-color: #000000;
}
.class2 {
text-decoration: underline;
}
.class3 {
letter-spacing: 0.5em;
}
.class4 {
text-transform: uppercase;
}
</style>
</head>

<body>
<strong>Пример 12<b/>
<p class="class1">Текст абзаца выравнивается по ширине, имеет отступ первой строки 20 пунктов и использует белый цвет символов на черном фоне.</p>
<p>В тексте абзаца <span class="class2">используется подчеркивание,
</span>
<span class="class3"> увеличение расстояния между символами</span>
```

, а также перевод символов в верхний регистр.**</p>**
</body></html>

На рис. 3.8 показано отображение текста примера 12 в браузере.

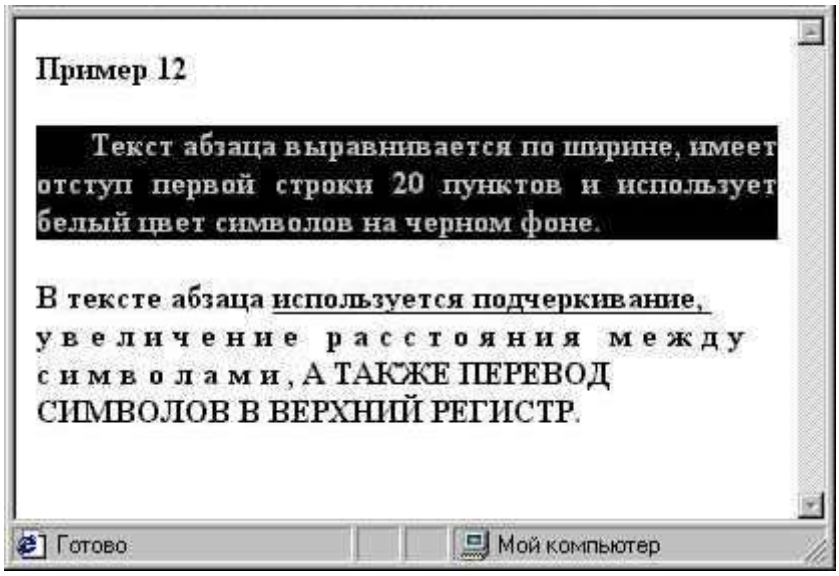


Рис. 3.8

3.5. Цвет и фон

1**. color** - цвет текста - любое значение, соответствующее стандарту (именованная константа (например, black установит черный цвет); с помощью RGB значения ('rgb(255,0,0)' установит красный цвет); с помощью шестнадцатеричного значения (например, #0000ff установит синий цвет)).

2**. background-color** - цвет фона - любое значение, соответствующее стандарту.

3. **background-image:URL("URL") -** фоновое изображение - любое значение URL, соответствующее стандарту.

4**. background-repeat -** направление повторения фонового изображения со значениями: **repeat** (повторение по горизонтали и вертикали - по умолчанию), **repeat-x** (повторение только по горизонтали), **repeat-y** (повторение только по вертикали), **no-repeat** (отсутствие повторения).

5**. background-position** - положение фонового изображения относительно верхнего левого угла содержащего его элемента со значениями: расстояние по горизонтали, расстояние по вертикали в стандартных единицах длины или в процентном соотношении. Значение 50% по горизонтали и 50% по вертикали дает расположение фонового изображения по центру. Размещение фонового изображения может быть также **top (**по верхнему краю),**center** (по центру), **bottom** (по нижнему краю), **left** (по левому) и **right** (правому) краям.

6**. background-attachment** - прокрутка фонового изображения вместе с документом со значениями: **scroll** (прокрутка - по умолчанию) или **fixed** (фиксация изображения в окне браузера).

7. Для описания сразу всех параметров фона можно использовать свойство **background: background-color background-image background-repeat background-attachment background-position**

В CSS3 добавлены следующие свойства.

8**.** Размер фоновых изображений может устанавливаться с помощью свойства **background-size.** Размер фоновых изображений может быть указан в пикселях или в процентах (первое значение указывает ширину изображения, а второе – высоту). Например,

background-image: url(images3.jpg); background-size:60% 30%;

9. CSS3 расширяет возможности свойства **background-image**. Один элемент может иметь несколько фоновых изображений одновременно. Для того чтобы вставить в элемент несколько фоновых картинок, необходимо перечислить пути к ним через запятую в свойстве **background-image**, например,

background-image:url(image1.gif),url(image3.jpg);
background-position:bottom right, center;
background-size:150px 40px,100% 100%;
background-repeat:no-repeat,no-repeat;. Изображения будут накладываться друг на друга в перечисленной очередности (т.е. первая заданная картинка будет отображаться поверх следующих). Для того, чтобы применить к фоновым изображениям свойства оформления, необходимо перечислить значения в нужном порядке через запятую (к примеру, в свойстве **background-position:bottom right, center; bottom right** будет применено к первой, а **center** ко второй фоновой картинке).

10. В CSS3 цвет может задаваться с помощью **HSL (Hue, Saturation, Lightness)** то есть тона, насыщенности и яркости.

Для того, чтобы задать цвет этим способом нужно указать:

1. **тон** - указывается в градусах поворота цветового круга – (*0 градусов - красный, 120 градусов - зеленый, 240 градусов - голубой и т.д.*);
2. **насыщенность цвета** - указывается в процентах (*по мере понижения значения цвет будет блекнуть*);
3. **яркость цвета** также указывается в процентах (*0% - темный, 100% - светлый*).

Задание цвета с помощью системы RGBA позволяет определять цвет и прозрачность одновременно.

Вначале необходимо указать значения RGB, а затем значение прозрачности (*0 - максимальная прозрачность, 1 - минимальная прозрачность*).

Задание прозрачности с помощью RGBA отличается от действия свойства **opacity** тем, что **opacity** делает прозрачным сам элемент и все его элементы-потомки, а RGBA делает прозрачным только сам элемент.

Например, **background-color:rgba(0,0,0,0.6); или opacity:0.6**;

11. Градиенты поддерживается во всех современных браузерах, но требуют добавления специального префикса. Для браузера IE10+ требуется префикс - ms, для Chrome и Safari префикс - webkit, для Opera префикс - o и для Firefox префикс - moz.

Линейные градиенты создаются с помощью CSS3-метода **linear-gradient**, который должен указываться в значение свойства **background**.

Для того чтобы создать линейный градиент, необходимо указать его направление (может задаваться с помощью ключевых слов или градусов) и цвета перехода, которые задаются в выбранных точках перехода, которых может быть несколько. Координаты точек перехода удобно задавать в процентах.

Например, **background:linear-gradient(top,white,green); или background:linear-gradient(left,white 0%,green 50%,red 70%, black 100%);**

С префиксом: **background:-webkit-linear-gradient(top,white,green);**

С помощью свойства **radial-gradient** можно задавать сферические градиенты, например,

background:radial-gradient(white 10%,blue 50%);

или background:radial-gradient(circle, red 0%,white 30%,black 60%);

Пример 13_1

```
<html><head><title>Пример 13_1 </title>
<style>
p {background-image: url(images3.jpg);
   background-position:center;
   border: 1px dotted gray;}
p.c1 {background-repeat: repeat-y;}
p.c2 {background-repeat: repeat-x;}
</style>
</head>

<body><strong>Пример 13_1</strong>
<p class="c1"> В данном абзаце фоновое изображение позиционировано по центру и распространяется по вертикали. В данном абзаце фоновое изображение позиционировано по центру и распространяется по вертикали. </p>
<p class="c2"> В данном абзаце фоновое изображение позиционировано по центру и распространяется по горизонтали. В данном
```

абзаце фоновое изображение позиционировано по центру и распространяется по горизонтали.**</p>**

</body></html>

На рис. 3.9 показано отображение текста примера 13_1 в браузере.

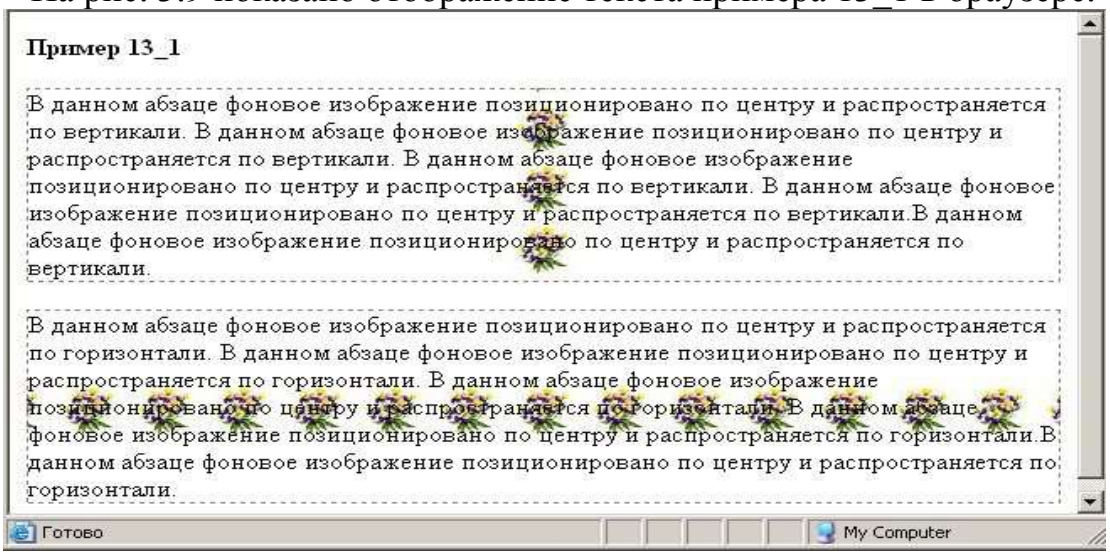


Рис. 3.9

В примере 13_2 фоновое изображение позиционировано на 25% по горизонтали и, по умолчанию, по центру по вертикали.

Пример 13_2

```
<html><head><title>Пример 13_2</title>
<style>
p {background-image: url(images3.jpg);
                background-position:25%;
                background-repeat: no-repeat;
                border: 2px dotted gray;}
</style>
</head>

<body>
<strong>Пример 13_2</strong><br>
<p>В данном абзаце фоновое изображение позиционировано на 25% по горизонтали и по центру по вертикали. В данном абзаце фоновое изображение позиционировано на 25% по горизонтали и по центру по вертикали. </p>
</body></html>
```

На рис. 3.10 показано отображение текста примера 13_2 в браузере.

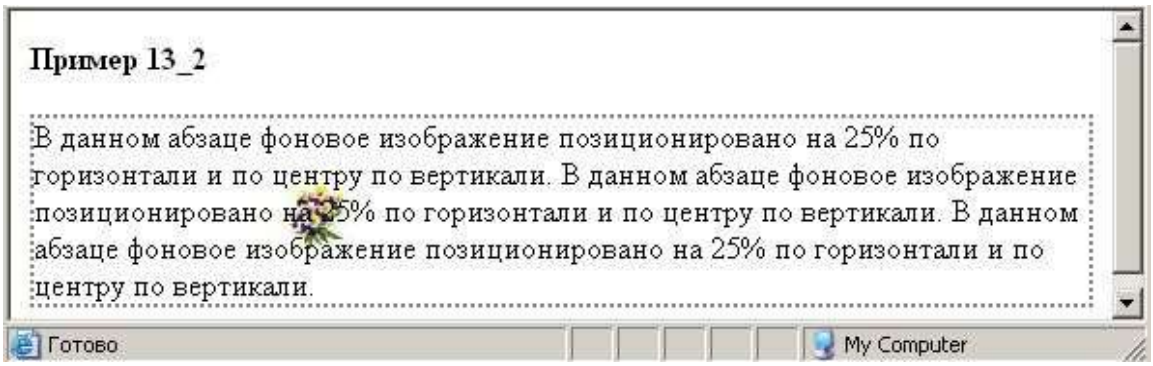


Рис. 3.10

3.6. Шрифты

1**. font-family** - семейство шрифтов. Может быть несколько семейств, отделенных друг от друга запятыми. Приоритет определяется порядком в списке. Значением может быть имя типового шрифта (**serif/sans-serif/cursive/fantasy/monospace**), а также шрифта, принадлежащего одному из типов.

2**. font-style** - начертание шрифта со значениями: **normal (**обычное по умолчанию), курсив (**italic)**, наклонное начертание (**oblique).**

3**. font-variant** - в виде малых прописных букв со значениями: **normal** (обычное по умолчанию) или **small-caps** (в виде малых прописных букв).

4**. font-weight** - толщина выводимого шрифта со значениями: **normal** (обычная по умолчанию), **bold** (полужирный), **bolder** (жирный), **lighter** (светлый) или числовыми значениями: 100-светлый, 400-обычный, 700-полужирный, 900-жирный.

5**. font-size** - высота символов (кегль). Значением является любая, соответствующая стандартам высота или процентное значение, обозначающее уменьшение или увеличение в процентах от кегля родительского элемента. Значения абсолютного размера могут быть записаны в виде ключевых слов: **xx-small** (крайне малый), **small** (малый), **medium** (средний по умолчанию), **large** (большой), **x-large** (очень большой), **xx-large** (крайне большой).

Значения относительного размера могут быть записаны в виде ключевых слов: **larger** (больше) и **smaller** (меньше).

Использование различных свойств шрифтов приведено в примере 14.

В CSS3 добавлены следующие свойства.

7**.** Использование подключаемых шрифтов. В предыдущих версиях CSS разработчики были вынуждены использовать только те шрифты, которые установлены на компьютере пользователя, в CSS3 можно использовать любые шрифты.

Необходимый шрифт размещается на сервере и подключается с помощью свойства **@font-face**.

Подключенный шрифт будет загружен и отображен автоматически при посещении страницы пользователем.

```
@font-face {
font-family:"opensans"; /* Имя шрифта */
src:url('opensans.woff') /* Местонахождение шрифта */
}
```

Браузеры IE9+, Chrome, Firefox, Opera и Safari поддерживают шрифты в формате .woff (Web Open Font Format - Открытый Формат Шрифтов Всемирной Паутины). Браузеры Chrome, Firefox, Opera и Safari также поддерживают шрифты в формате TTF и OTF, а IE в формате EOT.

Пример 14

```
<html><head><title>Пример 14 </title>
<style type="text/css">
p {
font: oblique  18pt Impact;}
.p1 {font: large Verdana, sans serif;
color:#0000ff;}
</style>

</head>

<body>
<strong>Пример14</strong>
<p>Для абзаца применен шрифт Impact стиля oblique, 18 пунктов</p>
<p class="p1">Для абзаца применен шрифт Verdana, large</p>
</body></html>
```

На рис. 3.11 показано отображение текста примера 14 в браузере.

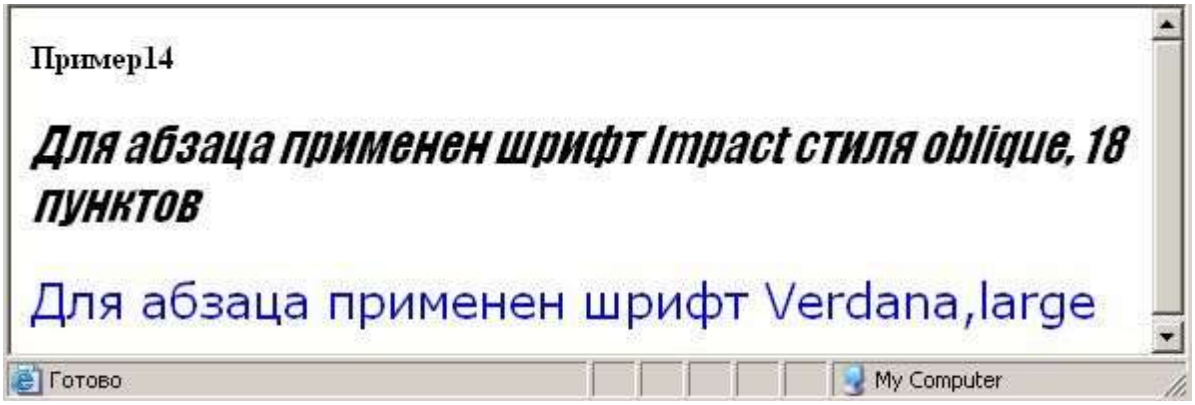
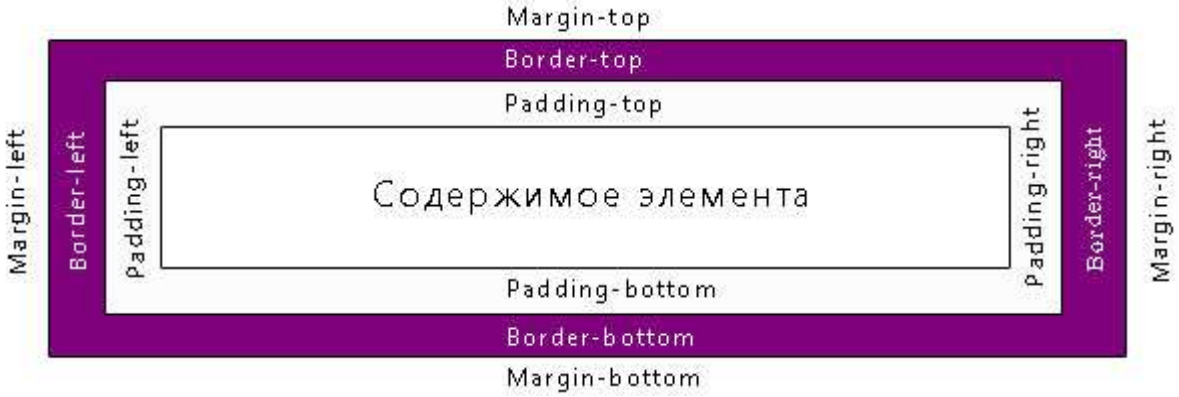


Рис. 3.11

3.7. Блочная модель

Все элементы в CSS являются прямоугольными блоками. Каждый такой блок имеет зону **content**, в которой располагается содержимое элемента (т.е. текст, изображения и т.д.). Вокруг зоны **content** могут располагаться необязательные зоны: **padding**, **border** и **margin**.



Зона **padding** окружает зону **content**. Данная зона используется для задания величины отступа содержимого элемента (зона **content**) от его границы (зона **border**). Зона может быть разбита на четыре части, которые могут оформляться независимо: **padding-top, padding-right, padding-bottom, padding-left.**

Зона **border** окружает зону **padding**. Данная зона позволяет задать элементу границу произвольной ширины, стиля и цвета. Зона может быть разбита на: **border-top, border-right, border-bottom, border-left.**

Зона **margin** окружает зону **border**. Данная зона позволяет задать величину внешнего отступа данного элемента от окружающих. Зона может быть разбита на: **margin-top, margin-right, margin-bottom, margin-left.**

1**. margin-top, margin-right, margin-bottom, margin-left** - ширина верхнего, правого, нижнего и левого поля. Значение по умолчанию – 0, значениями являются длины, соответствующие стандарту или процентное значение, определяющее отношение ширины поля к ширине элемента.

2. **margin** - ширина полей для всех сторон элемента. У этого свойства может быть от одного до четырех значений. Одно значение присваивается всем полям, из двух значений первое присваивается *верхнему* и *нижнему* полю, а второе - *левому* и *правому*. При трех: первое - верхнему полю, второе - левому и правому, а третье - нижнему.

3. **padding-top, padding-right, padding-bottom, padding-left** - ширина промежутка между содержимым элемента и определенным участком его границы. Значение по умолчанию – 0, значениями являются длины, соответствующие стандарту или процентное значение, определяющее отношение ширины промежутка к ширине элемента.

4. **padding** - ширина промежутка для всех сторон элемента. У этого свойства может быть от одного до четырех значений. Одно значение присваивается всем промежуткам, из двух значений первое присваивается *верхнему* и *нижнему* промежутку, а второе – *левому* и *правому*. При трех: первое - верхнему полю, второе – левому и правому, а третье – нижнему.

5. **border** - рамка. Для описания всех свойств рамки можно использовать это конструкцию: **border: border-width border-style border-color**.

Свойствами являются **border-width (**ширина рамки), **border-style** (стиль рамки) и **border-color** (цвет рамки).

5.1 **border-width** - ширина рамки. Значениями являются: **thin (**тонкая линия), **medium (**средняя – по умолчанию), **thick (**толстая), а также стандартные значения ширины.

Можно также устанавливать значения ширины рамки для определенной стороны. Для этого используются свойства:

border-top-width, border-right-width, border-bottom-width, border-left-width c аналогичными значениями**.**

5.2 **border-style** - стиль рамки. Значениями являются: **none (**отсутствие – по умолчанию), **hidden (**скрытая), **dotted (**пунктир), **solid (**сплошная), **double (**двойная), **dashed (**штрих-пунктир), **groove** (двойная борозда), **ridge** (гребень), **inset (**врезка**), outset (**орнамент).

Можно также устанавливать значения стиля рамки для определенной стороны. Для этого используются свойства:

border-top-style, border-right-style, border-bottom-style, border-left-style.

5.3 **border-color** - цвет рамки. Значением является любое, соответствующее стандарту.

Можно также устанавливать значения цвета рамки для определенной стороны. Для этого используются свойства:

border-top-color, border-right-color, border-bottom-color, border-left-color

В CSS3 добавлены следующие свойства.

6. С помощью нового CSS3 свойства **border-radius** можно делать углы элементов сглаженными, например, **border-radius:5px;.** Данное свойство может применяться не ко всем углам элемента, а только к определенным:

border-top-left-radius делает сглаженным только верхний левый угол элемента;

border-top-right-radius делает сглаженным только верхний правый угол элемента;

border-bottom-left-radius делает сглаженным только нижний левый угол элемента;

border-bottom-right-radius делает сглаженным только нижний правый угол элемента.

7. С помощью свойства **box-shadow** можно добавлять к элементам страницы тени.

Например,

box-shadow:4px 4px black;

box-shadow:6px 6px 6px 2px black;

box-shadow:0px 0px 6px 2px black inset;

В первом случае задаются смещения тени по горизонтали и вертикали и цвет тени. Во втором - добавляется радиус разброса тени и размер тени, а в третьем – указано, что тень "внутренняя".

Пример 15

```
<html><head><title>Пример 15</title>
<style type="text/css">
p  {
border: 10px double red;
margin: 50px 100px 100px 50px;
padding:20px }

</style>
</head>
<body>
<strong>Пример15</strong>
<p>Текст абзаца помещен в двойную рамку красного цвета, толщиной 10 пикселей. Установлены поля шириной 100 пикселей справа и снизу и 50 пикселей сверху и слева, а промежуток между текстом абзаца и рамкой установлен 20 пикселей со всех сторон.</p>
</body> </html>
```

На рис. 3.12 показано отображение текста примера 15 в браузере.

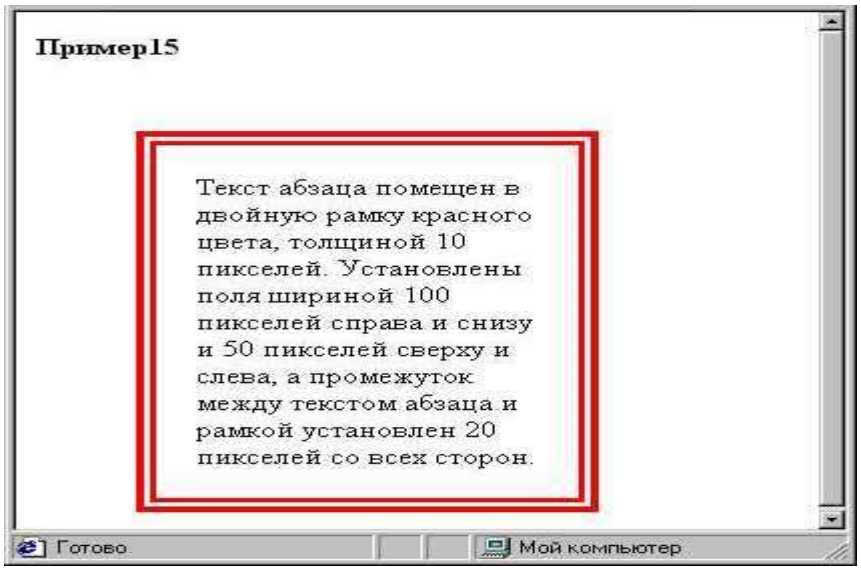


Рис. 3.12

3.8. Плавающая модель для размещения изображений и других элементов

До появления последних стандартов CSS для обтекания текстом изображений пользовались атрибутом **align** элемента **img** со значениями **left** и **right**. В настоящее время стандартизовано свойство **float** со значениями **left** и **right**, которое может быть применено не только к изображениям, но и к другим элементам HTML. В сочетании со свойством **float** часто применяют свойство **clear** со значениями **left** и **right** или **both** для того, чтобы освободить место слева или (и) справа от других элементов Web-страницы.

В примере 16_1 показано использование плавающей модели для изображения.

Пример 16_1

```
<html><head>      <title>Пример 16_1</title>
  <style>
  .leftFloat {float:left}
  .rightFloat {float:right}
        </style>
</head>

<body>
<strong>Пример 16_1</strong><br>
<img src="flax.gif" class="leftFloat" alt="flax" width="80px" height="80px" border="1px">
<p>Для обтекания текстом изображения слева  вместо атрибута align тега img используется свойство float со значением left. Для обтекания текстом изображения слева  атрибута align тега img используется свойство float со значением left. Для обтекания текстом изображения слева  вместо атрибута align тега img используется свойство float со значением left. </p>
<img src="flax.gif" class="rightFloat" alt="flax" width="80px" height="80px" border="1px">
<p >Для обтекания текстом изображения справа  вместо атрибута align тега img используется свойство float со значением right. Для обтекания текстом изображения справа  атрибута align тега img используется свойство float со значением right. Для обтекания текстом изображения справа атрибута align тега img используется свойство float со значением right.</p>
</body></html>
```

На рис. 3.13 показано отображение текста примера 16_1 в браузере.

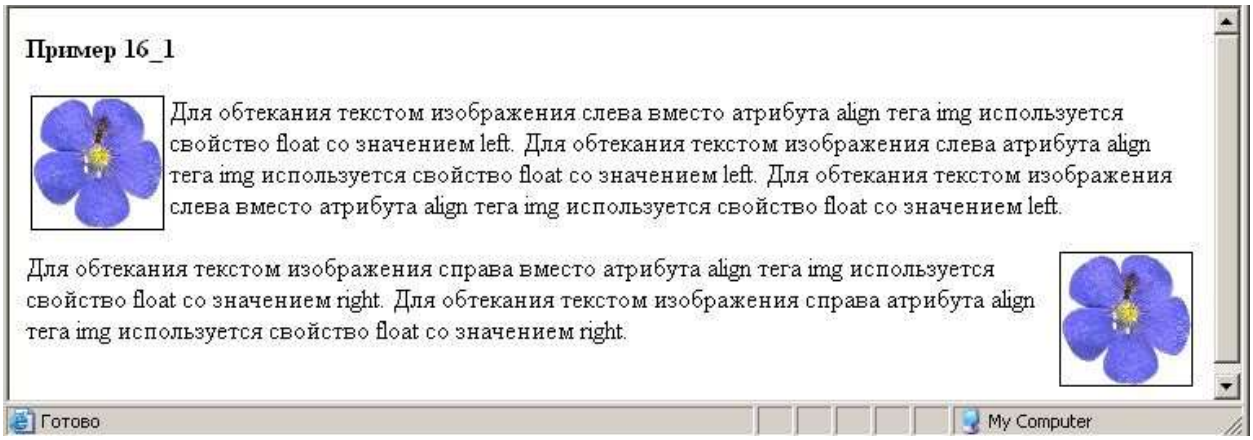


Рис. 3.13

В примере 16_2 показано использование плавающей модели для абзаца с цитатой.

Пример 16_2

```
<html><head><title>Пример 16_2</title>
<style>
p {font-family:Garamond;
text-align:justify;
margin-left:0.25em;}
.clearp {clear:left;}

 p.Floatl {float:left;
 width:45%;
 text-align:center;
 margin:0.5em 0.25em;
 padding:0.25em;
 border-top:1.3em  solid #999 ;
 border-bottom:1.3em  solid #999 ;
 background-color:black;
 color:white;
 font-size:1.3em;
 font-family:Garamond;
}
</style>
</head>

<body>
<strong>Пример 16_2</strong>
<p>В примере используется обтекание текстом абзаца цитаты. Свойства цитаты записаны в классе .Floatl</p>
```

** "**Фантазия важнее знаний" А.Эйнштейн ** **

** **Обтекание цитаты, заключенной в абзац, обеспечивается свойством float. Для обтекаемого элемента должна быть обязательно задана его ширина, кроме того в примере указаны свойства margin и padding, рамки серого цвета сверху и снизу, а также свойства шрифта: семейство, цвет символов, цвет фона и размер. Обтекание цитаты, заключенной в абзац, обеспечивается свойством float. Для обтекаемого элемента должна быть обязательно задана его ширина, кроме того в примере указаны свойства margin и padding, рамки серого цвета сверху и снизу, а также свойства шрифта: семейство, цвет символов, цвет фона и размер.** **

</body></html>

На рис. 3.14 показано отображение текста примера 16_2 в браузере.

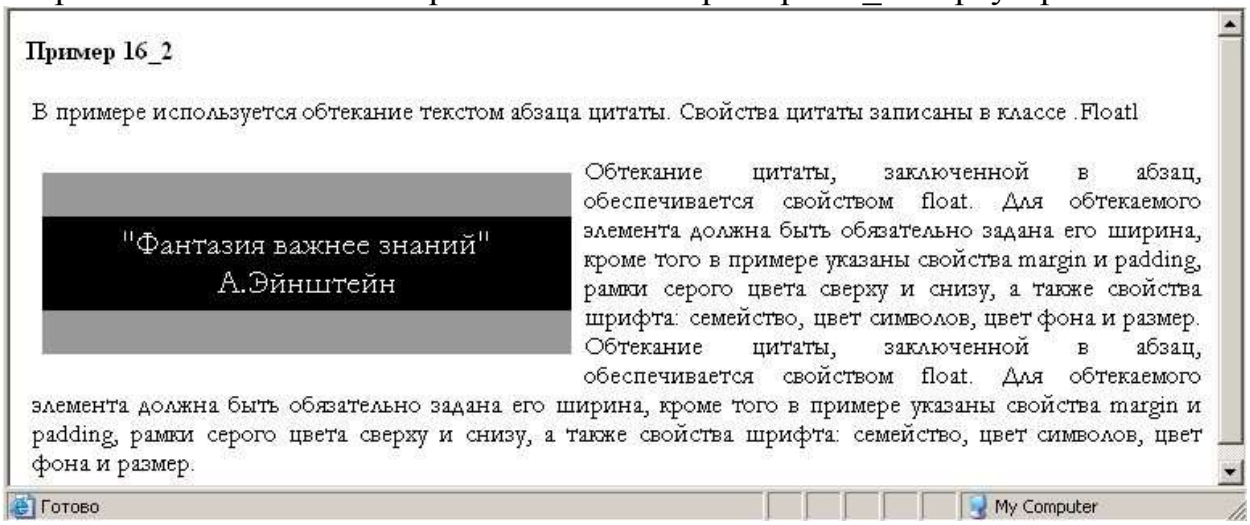


Рис. 3.14

В примере 16_3 показано использование обтекания для создания трехколонного документа с вставленной в одну из колонок цитатой.

Пример 16_3

```
<html><head><title>Пример 16_3</title>
<style type="text/css">
#header {color: white;
background-color: #666;
border-bottom:1px solid #333;
text-align: center;
padding: 0.1em 0;
margin: 0 0 1em 0}

#leftcol {width: 25%;
float:left;
```

```
background: white;
padding-bottom: 1em;
text-align: left;
}

#rightcol {
width:30%;
float: left;
background: white;
padding-left: 1em;
}

#centercol {
width:43%;
float: left;
background: white;
padding: 0 1em;
border-left: 1px solid #333;
border-right: 1px solid #333
}
#cite {
float:right;
width:90%;
border: 1px solid red;
padding:0.1em;
margin: 0 0.1em;
font-size:larger;
text-align:center;
}
#footer {
clear: both;
padding-bottom: 1em;
border-top: 1px solid #333;
text-align: center;}
</style>
</head>
<body>

<div id="header">
```

<h3>Технология каскадных таблиц стилей (Cascading Style Sheets, CSS)**</h3>**

** **

** **

** **Таблицы стилей - это набор элементов оформления, которые применяются к различным частям документа и описывают способ их представления на экране. В принципе, таблицы стилей реализованы во всех функционально-развитых текстовых процессорах.** **

** **

** **Принятие в 1996 году Консорциумом W3C CSS первого уровня в качестве стандарта позволило отделить содержание Web-страницы (текст, графические изображения и т.д.) от ее оформления (макет страницы и характеристики текста, например, шрифты, цветовое оформление и т.д.). После этого язык HTML вновь стал функционально-ориентированным, а не ориентированным на форму. Стандарт CSS2, принятый в 1998 году, основан на CSS первого уровня и позволяет разработчикам осуществлять контроль над Web-страницами на более высоком уровне. Он включает некоторые новые функции, в частности, возможность точно располагать элементы и объекты Web-страницы, применять загружаемые шрифты или использовать звуковые таблицы стилей. Применение стилей позволяет также решить многие проблемы поддержки браузеров, т.к. основные компании-разработчики браузеров встроили таблицы CSS в свои программные продукты.

** **

** **

** **С помощью таблиц стилей можно форматировать текст, используя методы, приближенные к методам форматирования обычных печатных страниц. Созданные страницы будут корректно функционировать, учитывая некоторые ограничения, связанные с платформами, браузерами, различными размерами экранов и разрешениями мониторов. Процесс разработки Web-стандартов в скором времени обещает решить эти проблемы.** **

** "**Искусство - это отражение мира в образах, а наука - отражение мира в понятиях" Н. К. ** **

** **Консорциум W3C CSS** **

</body></html>

На рис. 3.15 показано отображение текста примера 16_3 в браузере.

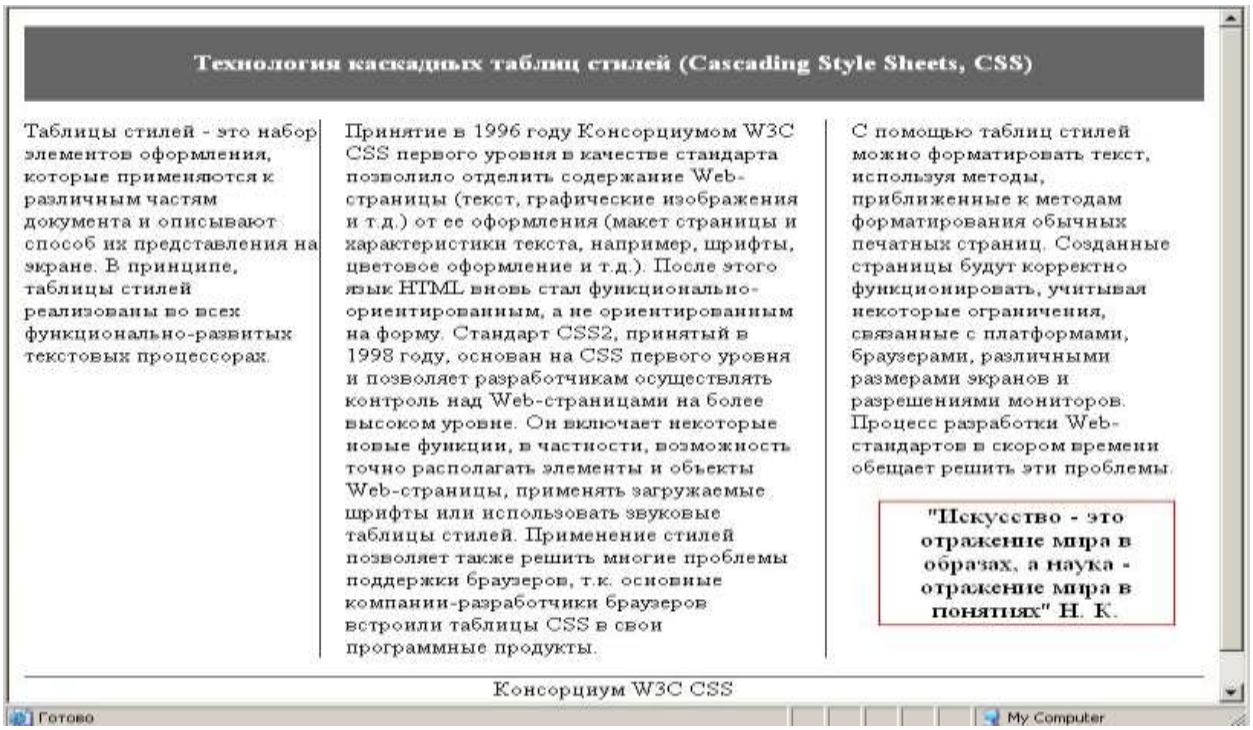


Рис. 3.15

3.9. Позиционирование и визуализация

Применение позиционирования предполагает использование ряда понятий:

Ограниченная область (блок-контейнер) – невидимая прямоугольная область, определенная браузером. Таблицы стилей позволяют управлять этой областью, устанавливая ее положение на странице с использованием абсолютных или относительных значений позиционирования.

Абсолютное позиционирование – технология, позволяющая задавать координаты блока относительно его блока–контейнера, которым может быть другой (родительский) элемент документа или начальный блок–контейнер – прямоугольник, соответствующий окну просмотра браузера. При этом блок элемента полностью удаляется из потока документа и позиционируется относительно его блока–контейнера.

Относительное позиционирование - технология, позволяющая задавать координаты блока относительно его несмещенного положения в потоке документа. При относительном позиционировании элемент сдвигается со своего обычного места, но пространство, которое он должен был занимать, не исчезает.

Фиксированное позиционирование – технология, позволяющая задавать координаты блока относительно начального блока – контейнера (окна просмотра браузера).

1. **position** - метод позиционирования блока, по умолчанию имеет значение **static**, другими значениями являются **absolute** (абсолютное) и **relative** (относительное) позиционирование, а также значение – **fixed**, при

котором позиционирование блока является смещением, как в случае абсолютного позиционирования, но блок фиксируется в окне браузера и не перемещается при прокрутке окна.

2. **top** - величина смещения вниз от верха его блока – контейнера, **bottom** – вверх от нижней стороны блока - контейнера, **left** – соответственно, влево, а **right** – вправо от левой и правой сторон блока - контейнера. Значениями являются любые, соответствующие стандарту длины, а также процентное значение: отношение в процентах длины смещения к ширине (высоте) блока, а также допустимы отрицательные значения указанных свойств смещения.

3. **width** - ширина блока. Значениями являются любые, соответствующие стандарту длины, а также процентное значение: отношение в процентах длины смещения к ширине окна.

4. **height** - высота блока. Значениями являются любые, соответствующие стандарту длины, а также процентное значение: отношение в процентах длины смещения к высоте окна.

5. z-index - z-индекс определяет порядок расположения блоков. Значениями являются целые числа (положительные и отрицательные), причем блоки с большими значениями z-индекса будут появляться над блоками с меньшими значениями.

6. **visibility** – видимость. Определяет, является ли элемент видимым - **visible** или скрытым - **hidden**.

7. **overflow** - управление переполнением. Имеет три значения: **visible** (элемент виден), **hidden** (перекрываемая часть отсекается), **scroll** (используется механизм прокрутки для визуализации элемента).

8. **clip: rect (top right bottom left) –** отсекание. Определяет вырезаемые области. Вырезаемая область определяется значениями сдвига соответственно сверху, справа, снизу и слева.

9. **display** – представление элемента со значениями **none**, **inline, block, list-item, table, inline-table** и другие значения, не рассматриваемые в данном пособии. (**display:none** скрывает элемент). Дополнительные сведения можно получить на сайте Консорциума WWW **http://www.w3.org/TR/CSS21/**

Пример 17_1 иллюстрирует основные варианты позиционирования.

```
<html><head><title>Пример 17_1</title>
<style  type="text/css">
p, h5 {margin:0;padding:0}
</style>
</head>

<body>
<strong>Пример 17_1</strong>
```

```
<div style="background-
color:'#c0c0c0';position:absolute;top:70;left:50;color:'#0000cc';">
<h5>relative</h5>
<p>Test text test text</p>
<p style="position:relative;top:70;left:50;color:'#cc0000'">
Test text test text</p>
</div>

<div style="background-
color:'#c0c0c0';position:absolute;top:70;left:250;color:'#0000cc'">
<h5>normal stream</h5>
<p>Test text test text</p>
<p style="color:'#cc0000'">Test text test text</p>
</div>

<div style="background-
color:'#c0c0c0';position:absolute;top:70;left:450;color:'#0000cc';>
<h5>absolute</h5>
<p>Test text test text</p>
<p style="position:absolute;top:70;left:50;color:'#cc0000'">Test text test
text</p>
</div>
</body></html>
```

На рис. 3.16 показано отображение текста примера 17_1 в браузере.

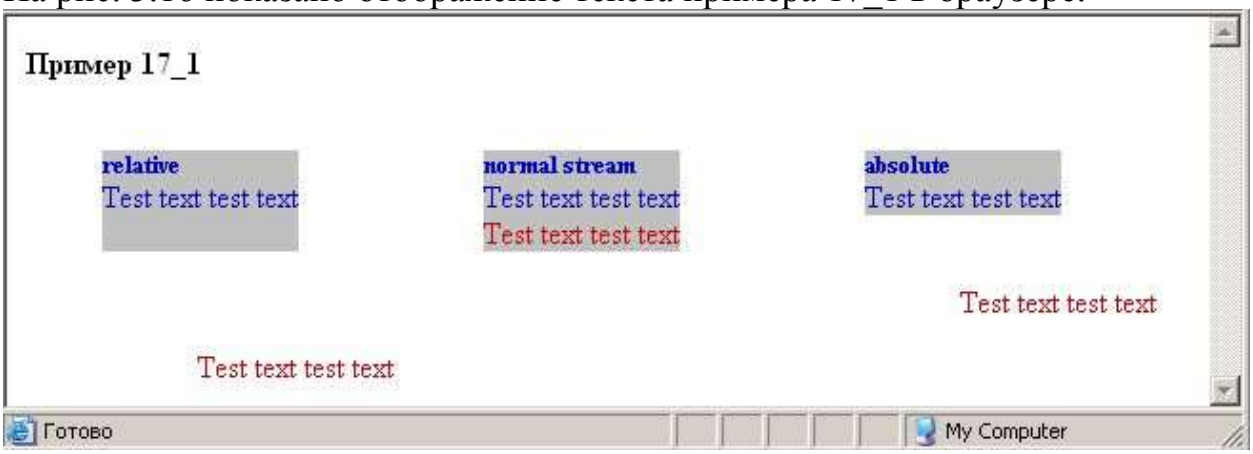


Рис. 3.16

В случае относительного позиционирования текст красного цвета отстоит на **top:70 и left:50** от своего несмещенного положения в потоке элементов, а в случае абсолютного – от родительского элемента, которым является слой **div**.

В примере 17_2 показано использование *абсолютного* позиционирования для отображения двух слоев. Вертикальный слой цвета

deeppink содержит логотип некоторого ЗАО, который позиционирован относительно его нормального положения в потоке элементов страницы, а также текст "ЗАО Воздушный шар", имеющий оформление, описанное в классе "**revers**".

Горизонтальный слой цвета **mistyrose** расположен поверх вертикального и содержит текст, оформленный согласно стилю **bluetext**.

Пример 17_2

```
<html><head><title>Пример 17_2</title>
<style type="text/css">
.revers {
font-weight:bold;
color:white;
text-align: center;}
.bluetext {
font-weight:bold;
color:darkblue;
text-align: justify;
padding:0.5em;
margin:0}
</style>
</head>

<body>
<strong>Пример 17_2</strong>
<!-вертикальный слой-->
<div style="position: absolute; top:0;
left: 250;width:200;
height:300;background:'deeppink'">

<img src="ball.jpg" style="position:relative; top:20;
left:25;" >
<p class="revers">ЗАО Воздушный шар</p>
</div>
<!-горизонтальный слой-->
<div style="position: absolute; top:155;
left:25;width:400;
height:90;background :'mistyrose' ">
<p class="bluetext">Слой цвета mistyrose с абзацем  лежит поверх правой колонки (слоя цвета deeppink) с логотипом. Изображение логотипа позиционировано относительно (relative) его нормального положения в потоке элементов страницы.</p>
```

```
</div>
</body>
</html>
```

На рис. 3.17 показано отображение текста примера 17_2 в браузере.

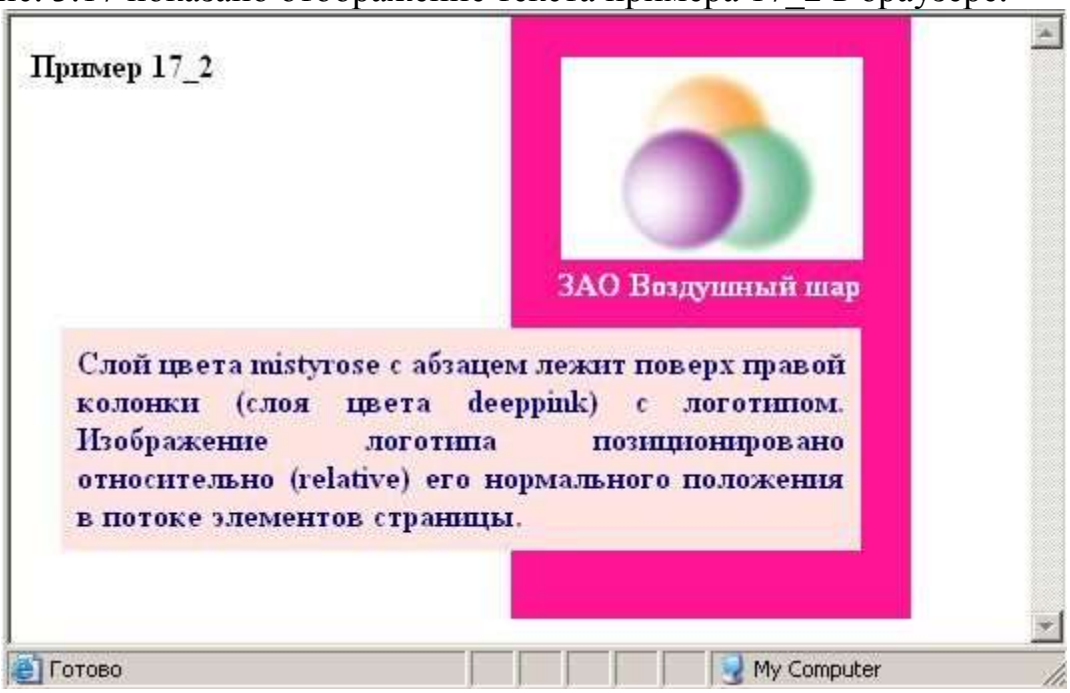


Рис. 3.17

Пример 17_3 демонстрирует использование свойства **z-index**, которое позволяет задать порядок (уровень) расположения элемента. Элемент ** ** используется для определения свойств слоя. Он является контейнерным элементом и позволяет применить позиционирование и **z-index** для одного или нескольких элементов, содержащихся в нем.

В примере 17_3 на web-странице выводится текст и слой ** ** серого цвета**.**

Пример 17_3

```
<html><head><title>Пример 17_3</title>
<style type="text/css">
<!--
.div_layer  {
background-color:#c0c0c0;
position:absolute;left:80;
 width:150; height:100;
visibility:visible;}
.div_text {
color:blue;
position:absolute;left:50;
width:350; height:100;
font-size:24pt;}
```

```
-->
</style>
</head>

<body>
<strong>Пример17_3</strong>
<div class="div_layer" style="top:50;z-index:1;">
</div>
<div class="div_text" style="top:50;"> z-индекс слоя больше, чем текста.
Слой расположен "ближе" к пользователю.</div>

<div class="div_layer" style="top:200;z-index:-1;">
</div>
<div class="div_text" style="top:200;"> z-индекс текста больше,чем слоя.
Текст расположен "ближе" к пользователю.</div>
</body></html>
```

На рис. 3.18 показано отображение текста примера 17_3 в браузере.

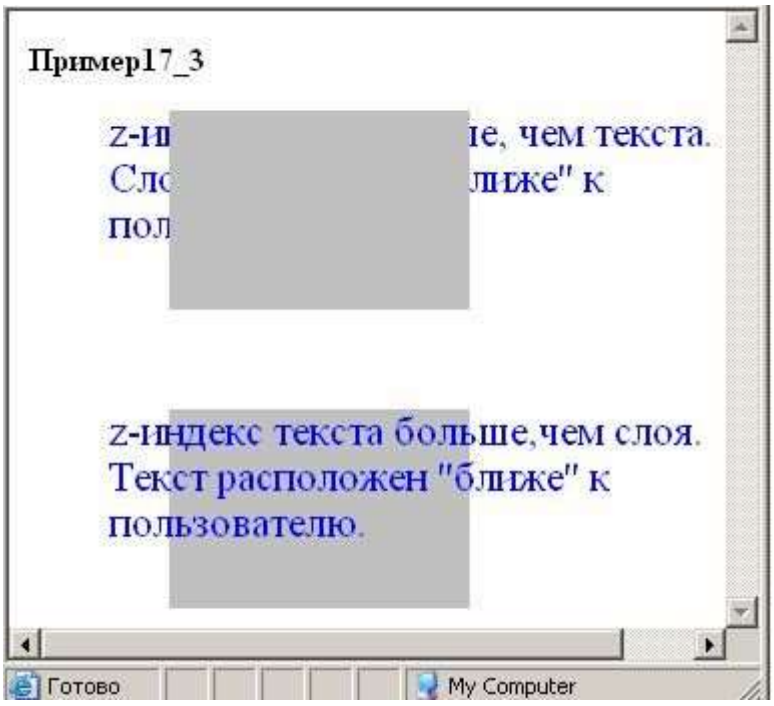


Рис. 3.18

Свойства слоя серого цвета и слоя текста описаны с помощью соответствующих классов в разделе описания стилей (<**style**>). В верхней части примера **z-index** серого слоя положителен (равен 1), тогда как текст имеет по умолчанию **z-index** равный 0. В результате слой текста находится на заднем плане, а серый слой – на переднем плане.

В нижней части примера **z-index** серого слоя отрицателен (равен –1), а текст лежит в нулевом слое. При этом текст оказывается на переднем плане, а серый слой – за текстом.

В примере 18 демонстрируется эффект, благодаря которому можно управлять представлением информации в пределах ограниченной области. Ограниченная область меньше, чем изображение, а переполнение скрыто. Таким образом, изображение "вписывается" в ограниченную область.

Область с переполнением помещена ниже обычного изображения, которое окружено тонкой рамкой, а область с переполнением – рамкой с большей толщиной.

Пример 18

```
<html>
<html><head><title>Пример 18</title>
<style type="text/css">
.overflow {
position: absolute; top:190;left: 50;
width:80;
height:80;
border:2px solid black;
overflow:hidden;
}
</style>
</head>

<body>
<strong>Пример18</strong><br>
<img  src="flower.gif" width=150 height=150 alt=”flower” border=1>
<img class="overflow" src="flower.gif" width=150 height=150 alt=”flower”>
</body>
</html>
```

На рис. 3.19 показано отображение текста примера 18 в браузере.

В примере 19 использован класс **clip1**, который описывает текст, помещенный в квадратную область, описанную дескриптором ** .** Класс **clip** обеспечивает применение к этой области отсекания (сверху и слева на 25 пикселей, а снизу и справа на 125 пикселей). На рис. 3.20 текст в квадратной области расположен справа, а отсеченный – слева.

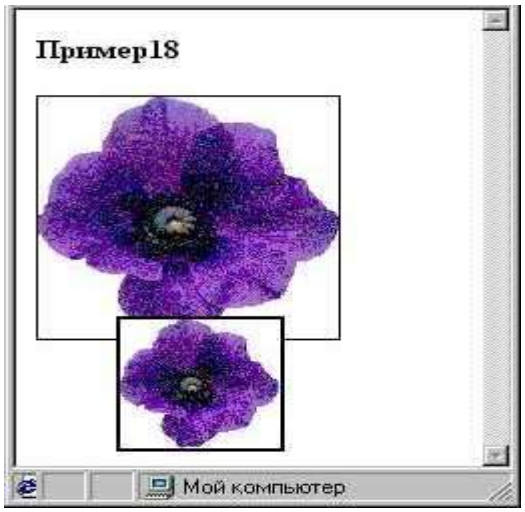


Рис. 3.19

Пример 19

```
<html><head><title>Пример 19</title>
<style type="text/css">
.clip {
position: absolute; top:50;
left: 230;
width:150;
height:150;
color:yellow;
background-color:black;
clip:rect(25px 125px 125px 25px);}
.clip1{
position: absolute; top:50;
left: 30;
width:150;
height:150;
color:yellow;
background-color:black;
}
</style>
</head>

<body>
<strong>Пример19</strong>
<div class="clip1">Текста абзаца будет обрезан при помощи отсекания, примененного к данной квадратной области размером 150 на 150 пикселей.</div>
```

** **Текста абзаца будет обрезан при помощи отсекания, примененного к данной квадратной области размером 150 на 150 пикселей.** **
</body></html>

На рис. 3.20 показано отображение текста примера 19 в браузере.

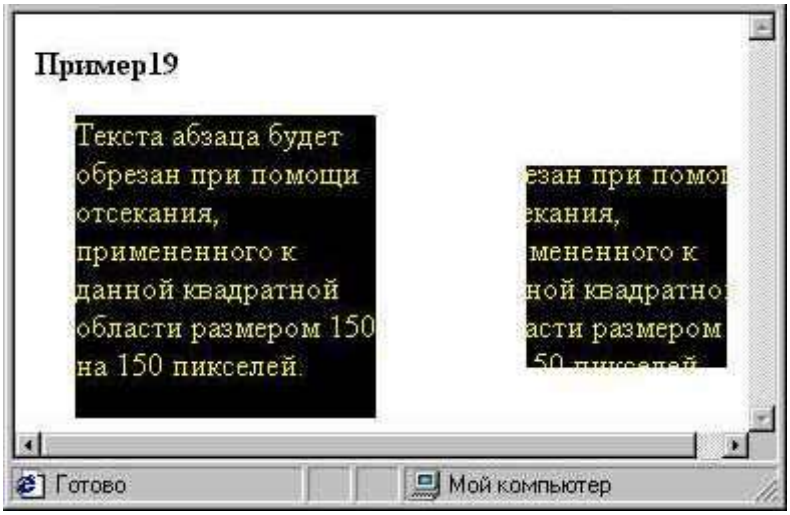


Рис. 3.20

В примере 19_1 показано применение свойства **display** для создания горизонтального меню из списка ссылок. *Блочный* элемент, которым является список ссылок, преобразован в *строчный* с оформлением, соответствующим элементам меню.

Пример 19_1

```
<html><head><title>Пример 19_1</title>
<style>
#nav h5 {text-align:center;font-size:16pt}
#nav ul {padding: 3px 0;
border-bottom:1px solid #778;
font :bold 12px Verdana, sans-serif;
}

#nav ul li {
display: inline;
}
#nav ul li a {
padding: 3px 0.5em;
margin-left: 3px;
border: 1px solid #778;
border-bottom: none;
background: #dde;
text-decoration: none;
```

```
}
#nav ul li a:link {
color: #0000ff;
}

#nav ul li a:visited {
color: #667;
}
 #nav ul li a:link:hover,  #nav ul li a:visited:hover {
color: #000;
background: #aae;
border-color: #227;}

#nav ul li a#current {
background: white;
border-bottom: 1px solid white;
}
  </style>

</head>
<body>
<div id="nav">
<h5> navigation bar <h5>
<ul>
<li> <a href="bg1.html">home</a></li>
<li> <a href="bg1.html">about</a></li>
<li> <a href="bg1.html">archives</a></li>
<li> <a href="bg1.html"  id="current">contact</a></li>
</ul>
</div>
</body></html>
```

На рис. 3.21 показано отображение текста примера 19_1 в браузере.

Рис. 3.21

3.10. Анимация в CSS3

Для создания анимации в CSS3 используется свойство **@keyframes**. Это свойство используется как контейнер, в который помещаются свойства оформления, подлежащие анимированию.

@keyframes имяАнимации
{
from {CSS свойства} /* Оформление элемента перед началом анимации */

to {CSS свойства} /* Оформление элемента после завершения анимации */
}

К элементу, который должен быть анимирован, необходимо добавить свойство **animation** и указать в нем *имя анимации* (1-е значение), *время в секундах*, в течение которого она будет выполняться (2-е значение), *количество повторов* анимации (3-е значение) и *направление анимации* с помощью значения alternate (4-е значение). При этом в нечетные разы анимация будет выполняться в заданном порядке, а в четные – в обратном. Если это значение не задано, то анимация выполняется в обычном порядке.

В примере 20.1 показана анимация в виде движущегося текста, помещенного в абзац. Абзац позиционирован и имеет координаты, определенные свойствами **left** и **top**, одна из которых (left) изменяется в ходе анимации. Анимация выполняется в течение 4-х секунд три раза и во-второй раз она выполняется в обратном порядке (от 500px до 25px) благодаря свойству **alternate**.

Пример 20_1

```
<html>
<style type='text/css'>

@keyframes anim{
from {left:25px;}
to {left:500px;}
}
@-moz-keyframes anim{
from {left:25px;}
to {left:500px;}
}
@-webkit-keyframes anim{
from {left:25px;}
to {left:500px;}
```

```
}
#par{
position:absolute;
top:100px;
left:25px;
font-size:16pt;
font-weight:bold;
animation:anim 4s 3 alternate;
-webkit-animation:anim 4s 3 alternate;
}
</style>
<body>
<p id="par">движущийся текст:</p>
</body>
</html>
```

Ход анимации можно определять не только с помощью ключевых слов **from** и **to**, но и с помощью %, что позволяет более точно контролировать ход выполнения анимации.

В примере 20_2 изменяется размер фонового изображения вначале в сторону уменьшения, а затем – в сторону увеличения. Ход выполнения анимации контролируется с помощью процентов.

Анимация происходит за 6 секунд один раз.

Пример 20_2

```
<html>
<style type='text/css'>
@keyframes anim {
0% {background-size:150px 250px;}
25% {background-size:75px 175px;}
50% {background-size:50px 150px;}
75% {background-size:75px 175px;}
100% {background-size:150px 250px;}
}
@-moz-keyframes anim {
0% {background-size:150px 250px;}
25% {background-size:75px 175px;}
50% {background-size:50px 150px;}
75% {background-size:75px 175px;}
100% {background-size:150px 250px;}
}
@-webkit-keyframes anim {
0% {background-size:150px 250px;}
```

```
25% {background-size:75px 175px;}
50% {background-size:50px 150px;}
75% {background-size:75px 175px;}
100% {background-size:150px 250px;}
}
#bgs {
background-image:url("bg.gif");
background-size:150px 250px;
animation:anim 6s 1;
-webkit-animation:anim 6s 1;
}
</style>
<body id="bgs">
</body>
</html>
```

3.11 Переходы CSS3

Свойство **transition** позволяет создать эффект перехода – изменение свойства элемента за определенное время. Для создания переходов необходимо указать CSS свойство, которое будет изменяться и скорость выполнения этих изменений в секундах. Для того, чтобы добавить эффект перехода к нескольким свойствам нужно перечислить их названия через запятую.

В примере 21 при наведении мыши на элемент <**div**> изменяются его свойства **background-color; color; width; height; font-size.**

Пример 21

```
<html>
<style type='text/css'>

#d1
{
background-color:#c0c0c0;
color:#F00;
width:300px;
height:100px;
font-size:16pt;
transition: color 3s, width 3s, background-color 3s;
-webkit-transition: color 3s, width 3s, background-color 3s;

}
#d1:hover
{
background-color:#f00;
color:#c0c0c0;
width:400px;
height:200px;
font-size:24pt;
}
</style>

<body>
<div id="d1">переход</div>
</body>
</html>
```

3.12 Трансформирование вCSS3

С помощью CSS3 свойства **transform** можно трансформировать элементы. В качестве значения данного свойства должна указываться одна из функций трансформирования, которые приведены в таблице.

Функция	Описание
translate(xpx,ypx)	Смещает элемент от изначальной позиции по горизонтали и вертикали.
translateX(x)	Смещает элемент по горизонтали.
translateY(y)	Смещает элемент по вертикали.
scale(x,y)	Растягивает элемент по вертикали и горизонтали.
scaleX(x)	Растягивает элемент по горизонтали.
scaleY(y)	Растягивает элемент по вертикали.
rotate(30deg)	Поворачивает элемент по часовой стрелке.
skew(x,y)	Скашивает элемент по горизонтали и вертикали.
skewX(x)	Скашивает элемент по горизонтали.
skewY(y)	Скашивает элемент по вертикали.
matrix(x,x,x,x,x,x)	Совмещает все перечисленные выше методы в один.

В примере 22 показано использование свойства **scale**, которое растянет элемент с идентификатором **#d1** в 2.5 раза по горизонтали и в 2 раза по вертикали.

Пример 22

```
#d1
{transform:scale(2.5, 2);}
```

Лекции 6,7

Глава 4. Объектные модели браузера (Browser Object Model, BOM) и документа (Document Object Model, DOM)

4.1 Объектная модель браузера (BOM, Browser Object Model)

Объектная модель браузера (BOM) – это инструмент, с помощью которого язык JavaScript получает доступ к объектам браузера.

Браузер интерпретирует html-страницы, но и страница может взаимодействовать с браузером благодаря тому, что браузер представлен в виде совокупности отдельных элементов – объектов. Объект *способен к изменению,* имеет *имя и* обладает *свойствами, методами и событиями*. Совокупность объектов браузера образуют объектную модель браузера (BOM).

BOM имеет иерархическую структуру, упрощенный вариант которой представлен на рис. 4.1.

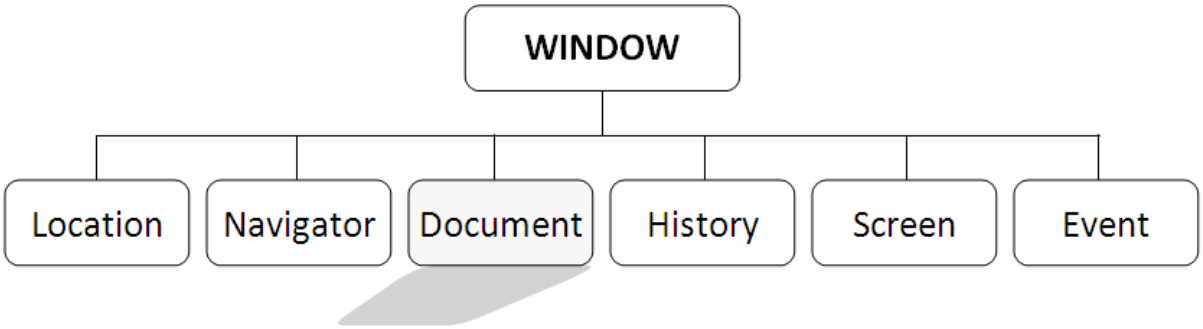


Рис. 4.1

Рассмотрим свойства и методы объектов BOM. Все свойства объекта могут быть выведены на экран с помощью кода JavaScript. К примеру, код для объекта **navigator** приведен далее:

```
var  option;
for (option in navigator){
document.write(option+":"+ navigator[option]+"<br/>");
}
```

4.1.1 Объект Window

Свойство **status** позволяет выводить или считывать информацию в строке состояния:

```
window.status=”сообщение в строке состояния”;
```

Метод **open** обеспечивает открытие файла в окне с заданными свойствами, указанными в аргументе **param**.

```
window.open("file1.html", "my_win", "param");
```

Метод возвращает ссылку на объект, сохраненную в примере в переменной **newWin**.

var newWin=window.open("file1.html", "my_win", "width=200, height=200");

Метод **alert** активизирует окно *предупреждения* с заданным сообщением:

window.alert("сообщение");

Метод **prompt** активизирует окно *приглашения* с заданным сообщением и текстовым полем для ввода информации пользователя:

window.prompt("сообщение", "умолчание");

Метод возвращает значение, введенное пользователем.

Метод **confirm** активизирует окно *подтверждения* с заданным сообщением и кнопками **ok** и **cancel**:

window.confirm("сообщение? ");

Метод возвращает значение **true** при нажатии кнопки **ok** и **false** при нажатии кнопки **cancel**.

Метод **setTimeout(функция, время)**обеспечивает запуск однократного таймера для отсчета интервала времени и запуска функции:

setTimeout(функция, время);

останов таймера:

t1= setTimeout(функция, время);

clearTimeout(t1);

Метод **setInterval(функция, время)** обеспечивает запуск таймера для периодического запуска функции через заданный интервал времени:

setInterval(функция, время);

останов таймера:

t2= setInterval(функция, время);

clearInterval(t2);

4.1.2 Объект location (адресная строка)

Свойство **href** сообщает адрес загруженной страницы, который находится в адресной строке:

Например, можно сохранить содержимое адресной строки и вывести на экран:

var a= location.href;

alert(a);

Свойство может также использоваться для записи:

location.href="file2.html";

4.1.3 Объект navigator

Объект содержит информацию об имени, версии, платформе, кодовом имени браузера, языке, подключению к Сети, наличию **cookie** и другую.

Свойство **appName** предоставляет информацию об имени браузера.

Свойство **appVersion** предоставляет информацию о версии браузера.

Например, можно вывести на экран значения свойств объекта navigator:

```
alert("имя браузера – "+navigator.appName+"версия="+navigator.appVersion);
```

Свойство **appCodeName** предоставляет информацию о кодовом имени браузера.

Свойство **userAgent** предоставляет наиболее полную информацию о браузере и используется браузером в заголовке http-протокола для передачи на сервер.

Свойство **language** предоставляет информацию о языке браузера, используемого пользователем.

Свойство **geolocation** предоставляет информацию о местоположении пользователя**.**

4.1.4 Объект history

Объект хранит адреса просмотренных страниц и позволяет реализовать действия, соответствующие действиям кнопок “назад” и “вперед”.

Метод **back()** позволяет загрузить предыдущую страницу ("назад").

Метод **forward()** позволяет загрузить следующую страницу из списка просмотренных (“вперед”).

Метод **go()** позволяет загрузить любую из просмотренных страниц.

4.1.5 Объект screen

Объект позволяет идентифицировать свойства монитора пользователя.

Свойство **colorDepth** предоставляет информацию о глубине цвета (числе бит на пиксел).

Свойства **width** и **height** предоставляют информацию о ширине и высоте экрана.

4.1.6 Объект event

Свойства объекта содержат информацию о событии.

Например, для события, генерируемого мышью, свойства **screenX** и **screenY** содержат X и У координаты курсора мыши относительно экрана.

4.1.7 Объект document

Свойство **lastModified** содержит дату последней модификации страницы.

Свойства **fgColor, bgColor** содержат информацию о цвете текста и фона.

Метод **write()** позволяет выводить информацию на экран.

Например,

document.write(document.lastModified); - выводится дата последней модификации страницы;

document.write(" content "); - создается html-код;

document.write("текст") - выводится текст.

4.2 Объектная модель документа (Document Object Model, DOM)

Объектная модель документа (DOM) – это инструмент, с помощью которого язык JavaScript получает доступ к документу, и может изменять содержимое html-страниц.

При загрузке документа браузер производит разбор его содержимого и на основе разбора создает объектную модель документа – DOM (Document Object Model).

DOM состоит из вложенных, иерархически расположенных *объектов*, которые называются узлами. Каждый узел в структуре представляет располагающийся на странице html элемент. Эту структуру называют также DOM-дерево.

Используя DOM можно взаимодействовать с содержимым html-документов (*считывать, изменять, удалять*), пользуясь конструкциями JavaScript без перезагрузки браузера.

Далее приведен код html-документа, а на рис. 4.2 - DOM, которая соответствует данному коду.

```
<html>
<head>
<title>DOM</title>
</head>
<body>
<h1>Пример DOM</h1>
<p id="par1" style="font-size:24pt">Текстовый контент</p>
</body>
</html>
```

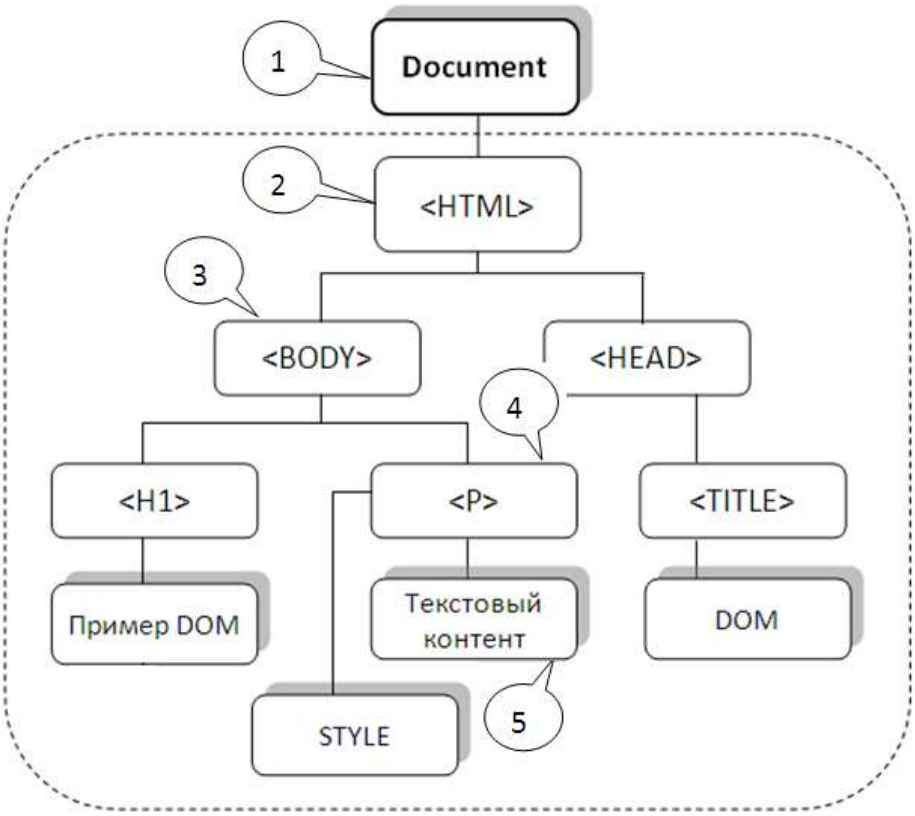


Рис. 4.2

Узлы (nodes) DOM различаются по своему назначению:

- узел HTML является *корневым*;
- элементам html соответствуют *элементные* узлы;
- содержимое элементов хранится в *текстовых* узлах;
- для атрибутов html-элемента создаются *атрибутные* узлы.

4.2.1 Отношения между узлами

Узлы в объектной структуре связаны друг с другом. Существует несколько специальных терминов для описания отношений между узлами.

Родительским узлом (**parent node**) по отношению к объекту является узел, в который вложен объект. На рис. 4.2 узел <**body**> является родительским по отношению к узлам <**h1**> и <**p**>. Для узла <**title**> родительским является узел <**head**>.

Узлом-*потомком* (**child node**) по отношению к объекту является узел, который вложен в объект. На рис. 4.2 узлы <**h1**> и <**p**> являются потомками по отношению к узлу <**body**>. Для узла <**head**> потомком является <**title**>.

Узлы-*братья* (**sibling node**) находятся на одинаковом уровне вложенности по отношению к их родительскому узлу. На рис. 4.2 узлами-братьями являются <**body**> и <**head**>, <**p**> и <**h**1>.

Самый верхний узел в DOM называется корневым. На рис. 4.2 корневым является узел <**html**> (объект **document** не является частью DOM).

4.2.2 Обращение к элементам

Для взаимодействия с элементами страницы необходимо определить, как происходит *обращение* к этим элементам.

Обращаться к элементам DOM можно следующими способами:

- по *идентификатору* элемента;
- по *имени* элемента;
- с помощью ключевого слова **this**
- перемещением по объектной структуре до необходимого элемента (рис. 4.2).

Для обращения к элементам DOM используются методы элементных узлов.

Метод объекта document **getElementById()** позволяет обратиться к элементу по его *идентификатору* (атрибуту **id**).

Если в условиях предыдущего примера элемент <**p**> имеет идентификатор **id="par1"**, то для обращения к нему используется код:

document.**getElementById**("**par1**");

Свойство элемета **innerHTML** позволяет *считывать* и *изменять* текстовое содержимое (контент) элемента:

document.**getElementById**("**par1**"). **innerHTML**;

document.**getElementById**("**par1**"). **innerHTML="новый текст"**;

С помощью метода **getElementsByTagName()** можно обратиться ко всем элементам с указанным *именем* тэга. Метод возвращает массив элементов. Нумерация элементов в массиве начинается с нуля.

Пусть в html-файле содержится два абзаца. Для обращения к контенту этих абзацев используется код:

```
document.getElementsByTagName("p")[0].innerHTML;
document.getElementsByTagName("p")[1].innerHTML ;
```

Сохраним результаты в переменных **rez** и **rez1** и выведем на экран с помощью конструкции **document.write**.

Количество элементов <**p**> может быть сосчитано с помощью свойства **length** массива этих элементов. Сохраним результат в переменной **rez2** и выведем на экран с помощью диалогового окна **alert**:

```
rez2= document.getElementsByTagName("p").length;
alert("Всего абзацев: "+rez2);
```

Код html-файла приведен далее.

```
<html>
<head>
<title>Пример getElementsByTagName()</title>
```

```
</head>
<body>
<p> Первый абзац </p>
<p> Второй абзац </p>
<script type='text/javascript'>
// содержимое первого абзаца сохраним в переменной rez
var rez=document.getElementsByTagName('p')[0].innerHTML;
//Выведем результат на экран
document.write(rez+'<br />');
// содержимое второго абзаца сохраним в переменной rez1
var rez1=document.getElementsByTagName('p')[1].innerHTML;
//Выведем результат на экран
document.write(rez1);
// количество элементов <p> сохраним в переменной rez2
rez2= document.getElementsByTagName('p').length;
//Выведем результат в окно предупреждения alert
alert("Всего абзацев: "+rez2);
</script>
</body>
</html>
```

Результат в окне браузера:

Первый абзац
Второй абзац
Всего абзацев: 2

DOM имеет методы, позволяющие *создавать* новые элементы, а также *вставлять* и *удалять* их из произвольных узлов.

С помощью метода **createElement()** можно *создать* элемент.

С помощью метода **appendChild()** можно *вставить* созданный элемент в произвольный узел.

С помощью метода **removeChild()** можно *удалить* узел - потомок из элемента.

Создадим новый абзац и вставим его в элемент <**body>.**

Код html-файла приведен далее.

```
<html>
<head>
<title>Пример createElement()</title>
</head>
<body>
<script type='text/javascript'>
//Создадим новый абзац
```

```
var newpar = document.createElement('p');
//Запишем созданный абзац в body
document.body.appendChild(newpar).innerHTML='<b>новый элемент </b>';
</script>
</body>
</html>
```

Результат в окне браузера:

новый элемент

Используем *последовательное перемещение* по объектной модели (рис. 4.2) для того чтобы найти текстовое содержимое узла <**p**> файла, DOM которого приведена на этом рисунке. Для этого воспользуемся некоторыми свойствами элементов.

С помощью свойства **childNodes** можно обратиться к узлам - потомкам элемента, которые образуют массив, индекс 0 в нем соответствует первому узлу – потомку, 1 – второму и т.д.

Для того чтобы обратиться к первому узлу - потомку используется свойство **childNodes[0], ко второму –childNodes[1] и т.д.**

С помощью свойства **nodeValue** можно найти контент (содержимое) узла.

Последовательное перемещение по объектной модели:

1. Обращение к объекту **document**, в котором находится DOM. Код на данном шаге имеет вид: **document.**
2. Обращение к корневому узлу (т.е. тэгу <**html**>), который находится внутри объекта **document**. Код на данном шаге имеет вид: **document.documentElement.**
3. Обращение ко второму потомку корневого узла, поскольку в коде страницы **body** располагается после **head**. Код на данном шаге будет иметь вид: **document.documentElement.childNodes[1].** Можно напрямую обратиться к **body**, используя следующий код **document.body**.
4. Обращение ко второму потомку **body** (элемент <**p**> задан в коде после <**h1**>). Код на данном шаге будет иметь вид: **document.body.childNodes[1]**.
5. Обращение к текстовому узлу, который является первым потомком <**p**> и его свойству. Код на данном шаге будет иметь вид: **document.body.childNodes[1].childNodes[0].nodeValue**.

С помощью свойства **nodeName** можно найти имя узла.

В условиях предыдущего примера для определения имени узла абзаца (**p**) нужно использовать код:

document.getElementById("par1").nodeName;

Для вывода на экран сохраним результат в переменной **rez** и используем конструкцию **document.write**.

Код html-файла приведен далее.

```
<html>
<head>
<title>Пример nodeName</title>
</head>
<body>
<p id='par1'> Абзац с id=par1 </p>
<script type='text/javascript'>
//Найдем имя элемента с id=par1 и сохраним результат в переменной rez
rez=document.getElementById('par1').nodeName;
//Выведем результат на экран
document.write(rez+'<br />');
</body>
</html>
```

Результат в окне браузера:

p

С помощью свойства **parentNode** можно обратиться к родительскому узлу элемента.

В условиях предыдущего примера для определения имени родительского узла элемента нужно использовать код:

document.**getElementById**("**par1**").**parentNode**.**nodeName**;

Для вывода на экран сохраним результат в переменной **rez** и используем конструкцию **document.write**.

Код html-файла приведен далее.

```
<html>
<head>
<title>Пример parentNode </title>
</head>
<body>
<p id='par1'> Абзац с id=par1</p>
<b>
<script type='text/javascript'>
//Найдем имя родительского узла элемента с id=par1 и сохраним результат в rez
rez=document.getElementById('par1').parentNode.nodeName;
//Выведем результат на экран
document.write(rez+'<br/>');
```

```
</script>
</b>
</body>
</html>
```

Результат в окне браузера:

Body

С помощью свойства **attributes** можно обратиться к атрибутам узла, что позволит изменить или считать значение атрибута.

Для того чтобы обратиться к первому атрибуту используется конструкция **attributes[0],** к следущему - **attributes[1]** и т.д.

В условиях предыдущего примера добавим в элемент <**p**> атрибут **style="font-size:24pt"** Тогда элемент будет иметь вид:

```
<p  id="par1" style="font-size:24pt">Абзац с id=par1</p>
```

Для определения значения первого атрибута элемента <**p>** с **id=par1** нужно использовать код:

```
document.getElementById('par1'). attributes[0].nodeValue
```

Для определения значения последнего атрибута элемента **<p>** с **id=par1** нужно использовать код:

```
document.getElementById('par1'). attributes[1].nodeValue
```

Для вывода на экран сохраним результат в переменных **rez** и **rez1** и используем конструкцию **document.write**.

Код html-файла приведен далее.

```
<html>
<head>
<title>Пример attributes  </title>
</head>
<body>
<p id='par1' style='font-size:24pt'>Абзац с id=par1</p>
<b>
<script type='text/javascript'>
//Значение первого атрибута
rez= document.getElementById('par1').attributes[0].nodeValue;
//Выведем результат на экран
document.write(rez+"<br />");
//Значение последнего атрибута
rez1= document.getElementById('par1').attributes[1].nodeValue;
//Выведем результат на экран
document.write(rez1+"<br />");
</script>
</b>
```

```
</body>
</html>
```

Результат в окне браузера:

par1
font-size:24pt

4.2.3 Изменение свойств CSS

Свойства элемента, заданные с помощью оформительских возможностей CSS, можно изменять с помощью объекта **style**.

В стилевых свойствах, написание которых включает дефис, например, **background-color**, свойство записывается без дефиса, а часть после дефиса начинается с прописной буквы - **backgroundColor**.

В приведенном далее коде строки таблицы приобретают цвет фона после выполнения кода JavaScript.

```
<html>
<head>
<title>Пример style css  </title>
</head>
<body>
<table width=30%>
<tr id='r11'>
  <td>1</td>
  <td>2</td>
</tr>
<tr id='r12'>
  <td>3</td>
  <td>4</td>
</tr>
</table>
<script type='text/javascript'>
//Установим желтый цвет фона для элемента r11
document.getElementById('r11').style.backgroundColor='yellow';
//Установим красный цвет фона для элемента r12
document.getElementById('r12').style.backgroundColor='red';
</script>
</body>
</html
```

4.2.4 События

Изменение элемента (объекта) происходит с помощью кода JavaScript, как было показано в предыдущем примере, или в момент наступления *события*, которое может инициировать пользователь или программное обеспечение (в частности, браузер).

События обрабатываются обработчиком события (**event – handler**). Чаще всего обработчиком события является функция JavaScript или программный код небольшого объема.

Список наиболее часто происходящих событий приведен далее:

click (щелчок мыши)	**mouseout** (сведение мыши с элемента)	**change** (выбор элемента списка)
dblclick (двойной щелчок мыши)	**load** (загрузка страницы)	**focus** (курсор на элементе)
mouseover (наведение мыши на элемент)	**unload** (выгрузка страницы)	**blur** (курсор вне элемента)

Средством подключения обработчика события в html является атрибут события. Атрибут события формируется из названия события и префикса on:

onclick	**onmouseout**	**onchange**
ondblclick	**onload**	**onfocus**
onmouseover	**onunload**	**onblur**

Например, по щелчку на кнопке на экране должно отобразиться сообщение о последней модификации страницы. Код приведен далее.

```
<form>
<input type=button value="Last" onclick="document.write('Last Modified was '+document.lastModified) "
</form>
```

Обработчик события обычно представлен с помощью функции:

```
<html>
<head>
<script type="text/javascript">
function LastMod(){
  document.write("Last Modified was "+document.lastModified);
}
</script>
</head>
```

```
<body>
<input type="button" value="Last"  onclick="LastMod ()"/>
</body>
</html>
```

Если обращение к объекту происходит из самого объекта, то для обращения к объекту можно использовать ключевое слово **this**. Например, при создании ролловера, в котором изображение меняется при наведении мыши, и возвращается в прежнее состояние при сведении мыши с объекта.

Это достигается изменением свойства **src** (**source** – источник) объекта **img** при наступлении событий **mouseover** и **mouseout**. Обращение к объекту происходит из самого объекта с помощью ключевого слова **this**:

```
<img src="cat.gif" alt="cat" width=100 height=100
onmouseover="this.src='dog.gif'" onmouseout="this.src='cat.gif'"
```

Рассмотрим пример *изменения фона* слоя при щелчке на кнопке. При этом должно изменяться стилевое свойство **background-color**.

Для решения этой задачи создадим три файла:

myFile.html с html-кодом,

myStyle.css со стилевыми свойствами,

myScript.js с кодом JavaScript.

Файл **myStyle.css** подключается к файлу **myFile.html** с помощью элемента

<link rel="stylesheet" href="myStyle.css" type="text/css"> в элементе **<head>**.

Файл **myScript.js** подключается к файлу **myFile.html** с помощью элемента

<script src="myScript.js" type="text/ javascript"> в части <body>, когда DOM-дерево сформировано, и можно использовать его объекты.

Код файла **myFile.html** приведен далее.

```
<html>
<head>
<link rel="stylesheet" href="myStyle.css" type="text/css">
</head>
<body>
<div id='d1'>
</div>
<form>
<input type="button" id="but" value="change bg">
</form>
<script  src="myScript.js" type="text/javascript"> </script>
</body>
</html>
```

Код файла **myStyle.css** приведен далее.

```
div {
background-color:red;
position:absolute;
left:50px;
top:50px;
width:200px;
height:200px;
}
```

Код файла **myScript.js** приведен далее.

```
function start(){
document.getElementById("d1").style.backgroundColor="green";
}
var divBg=document.getElementById("but");
divBg.onclick=start;
```

В переменной **divBg** сохранено обращение к элементу <**button**> (кнопка) с **id="but".**

```
var divBg=document.getElementById("but");
```

Следующая строка обеспечивает подключение обработчика события (функции **start**) в ответ на произошедшее в элементе **<button>** (кнопке) событии **click**.

```
divBg.onclick=start;
```

Лекции 8, 9, 10

Глава 5. Технологии создания web-страниц. Язык сценариев JavaScript

В 1995 году фирмой Netscape для своего браузера был разработан язык создания сценариев JavaScript, который первоначально назывался LiveScript. Его создание связывают с именем Брендана Эйха. В разработке JavaScript принимала участие и компания Sun, которая является автором языка Java, что и послужило причиной для выбора названия JavaScript. Его синтаксис и семантика должны были быть тесно связаны с Java, ожидалось также, что этот язык сценариев выступит в качестве альтернативы большому числу CGI сценариев и позволит загруженной в браузер странице динамически управлять своим содержимым и собственно браузером.

Возможности первой версии JavaScript были крайне ограничены и позволяли управлять свойствами только нескольких дескрипторов. Первым браузером, поддерживающим JavaScript, был Netscape Navigator 2.0.

Выработкой единого стандарта занялась Европейская ассоциация производителей компьютеров (European Computer Manufacturing Association, ECMA) в 1996 году.

Язык JavaScript в сочетании с каскадными листами стилей (CSS) и объектной моделью документа (DOM) позволяет полностью контролировать все, что происходит на web-странице.

Перечислим основные свойства языка JavaScript.

JavaScript является языком сценариев (script), это означает, что команды JavaScript интерпретируются браузером, а не компилируются в бинарный код. Недостаток интерпретируемого языка заключается в больших затратах времени на интерпретацию. Преимуществом же является легкость и быстрота изменения исходного кода, а также отсутствие дорогостоящих компиляторов.

JavaScript является объектно-базированным языком, объектная модель JavaScript основывается на объектах-экземплярах, а не на концепции наследования и не позволяет программисту создать собственный класс объектов, что является свойством объектно-ориентированного программирования (ООП).

Большинство созданных кодов на JavaScript реагируют на события, генерируемые пользователем или системой. Язык JavaScript обладает поддержкой обработки событий.

JavaScript является слабо типизированным языком в отличии от языков программирования, таких как Java или C++, требующих объявления типов переменных, используемых в программе.

Наконец, еще одной особенностью JavaScript является возможность внедрения кода в HTML-документ. Большинство объектов JavaScript

представлено соответствующими дескрипторами HTML. Код JavaScript может быть внедрен как в часть **<head>**, так и непосредственно в тело html-документа.

JavaScript применяется как в клиентской части, так и на стороне сервера (в частности, в ASP).

Инструментом для написания сценариев может служить простой текстовый редактор (Notepad) или средства визуального проектирования, такие как Notepad++, PHP Designer и другие.

5.1. HTML и JavaScript

HTML и JavaScript – независимые друг от друга технологии и существуют специальные правила помещения JavaScript на web-страницу.

Первый способ заключается в том, что сценарий JavaScript вставляется блоком в элемент **<head>.** Этот блок ограничивается парными дескрипторами
<script type="text/javascript"> и **</script>.** Назначением этого блока является указание браузеру на то, что содержимым является сценарий, который требует соответствующей обработки. Помещенный в блок **<head>** сценарий будет выполнен в первую очередь.

Сценарий может быть расположен не только в дескрипторе **<head>,** но и встроен непосредственно в html-код. В этом случае сценарий должен выполняться *после* создания браузером DOM-дерева.

Кроме указанных способов размещения код **JavaScript** рекомендуется располагать во внешнем файле (с расширением .**js**), тогда в дескрипторе **<script>** нужно использовать атрибут **src**, который будет указывать на этот файл.
<script type="text/javascript" src="my.js"> </script>. Этот вариант присоединения удобен в тех случаях, когда код должен использоваться несколькими html-страницами.

5.2. Основы программирования на JavaScript

5.2.1. Типы данных

В языке JavaScript различают следующие типы данных:

- строковые;
- числовые;
- булевы.

В пределах типа данные могут быть разделены на *литералы* и *переменные*. *Литералы* представляют собой постоянные значения JavaScript. Рассмотрим разные виды литералов.

Строка представляет собой набор символов, включающий в себя буквы, цифры, знаки пунктуации, заключенный в двойные или одинарные кавычки. Например,

"Я изучаю JavaScript"
"А Вы изучаете JavaScript?"
""

В последнем случае строка не содержит символов, в этом случае она называется "пустой" строкой.

Двойные кавычки могут содержаться внутри строки, заключенной в одинарные кавычки и наоборот. Например, текст JavaScript вставленный в html-документ:

OnLoad="function_1('arg1')".

Числа различаются *целые* и с *плавающий точкой.*

Целые числа включают в себя положительные, отрицательные числа и ноль, например, 10, -10, 0.

Числа с плавающей точкой содержат дробную часть, они могут быть представлены в экспоненциальном виде, например, 3141592.6538 или 3.141592e+6.

Второе число при этом равно 10 в шестой степени, умноженному на 3.141592. В результате получим 3141592.

Диапазон используемых чисел находится в пределах от 10^{308} до 10^{-308}.

Булевы литералы имеют только два значения **true** и **false**. Они используются в выражениях и условиях, например, выражение **if (pass= ="")** может принимать значения **true** или **false** в зависимости от выполнения или невыполнения условия.

Управляющие последовательности применяются для отображения в сценарии специальных знаков или их комбинаций. Они состоят из символа наклонной черты и управляющего кода, например,

\b-backspace;
\n-новая строка
\f-новая страница
\t-табуляция
\r-возврат каретки
\'-одинарная кавычка
\"-двойная кавычка
\\-наклонная черта

Например, код сценария

document.write("\"Фантазия важнее знаний.\" ");
document.write("Альберт Энштейн");

обеспечит вывод на экран строку:

"Фантазия важнее знаний." Альберт Энштейн

К особым типам данных можно отнести *несуществующее число* (not a namber-**NaN**), *неопределенный* (**null**) и *неопределяемый* (**undefined**) тип.

Под несуществующим числом понимается результат ошибочной математической операции (деление на ноль). Неопределенный тип имеет

одно значение – **null** и свидетельствует об отсутствии полезной информации. Неопределяемый тип имеет место при использовании неопределенных переменных и в ряде других случаев, приводящих к невозможности выполнения сценария.

Переменные позволяют хранить данные. *Под переменной понимается имя, присваиваемое ячейке памяти компьютера, хранящей определенные данные во время выполнения сценария*. До использования переменных в сценарии их надо *объявить* и *назначить* им определенное значение (*инициализировать*).

Для объявления переменных используется ключевое (зарезервированное) слово **var** (**variable**), за которым следует имя переменной. При выборе имени нужно помнить следующие правила:

- первым символом должна быть буква или символ подчеркивания;
- в имени недопустимы пробелы, знаки пунктуации и ключевые (зарезервированные) слова JavaScript**;**
- можно использовать символы верхнего и нижнего регистров, но учитывать тот факт, что JavaScript *чувствителен* к *регистру*;.
- длина может быть любой.

Например, для объявления переменной **msg** нужно использовать конструкцию:

var msg;

Несколько переменных можно объявить в одной строке через запятую:

var msg, msg1,msg2;

Значение переменной может быть присвоено в любом месте сценария. Для этого применяется оператор присваивания, в левой части которого стоит имя переменной, а в правой - присваиваемое значение, например,

msg="переменная JavaScript";

Переменной может быть присвоено значение в момент ее объявления, например,

var msg="переменная JavaScript", msg1="также переменная JavaScript";

Далее приведен пример использования переменной в сценарии. Переменная **name** объявлена, а затем инициализирована с помощью метода **prompt().** Имя пользователя, которое он введет в диалоговом окне, будет присвоено переменной **name**, а затем она будет выведена с помощью метода **alert** в окне вывода сообщений.

Пример 1

```
var name;
name=prompt("Как Вас зовут?", "пожалуйста, введите Ваше имя");
alert ("привет, "+name);
```

5.2.2. Выражения

Преобразование данных производится с помощью *выражений*. Выражение представляет собой совокупность *операндов*, над которыми производятся *операции.* Результатом выражения является значение, которое относится к одному из известных в JavaScript типу данных.

5.2.2.1. Операции

Арифметические операции

Арифметические операции служат для получения числовых результатов. Их перечень приведен в таблице 5.1.

Таблица 5.1

Операция	Название	Пример	Результат
+	Сложение	5+2	7
-	Вычитание	5-2	3
/	Деление	5/2	2.5
*	Умножение	5*2	10
++	Инкремент	5++	6
--	Декремент	5--	4
%	Целочисленное деление	5%2	1
-	Отрицание	-5	-5

Для присваивания значения переменным часто используется операция присваивания, например, **x=25**. Данная операция обеспечивает присваивание переменной **x** значения **25**. Когда JavaScript встречает операцию присваивания, он сначала анализирует правую часть выражения с целью определения значения, а затем – левую часть, которая содержит переменную, предназначенную для хранения полученного значения.

Арифметические операции используются в сочетании с операцией присваивания, что позволяет присваивать переменной новые значения, например, **x=5+7** или **x=x+y**. Запись комбинации операции присваивания и арифметических операций производится так, как показано в таблице 5.2.

Таблица 5.2

Сокращенная запись	Полная запись
x+=y	x=x+y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y
x%=y	x=x%y

В программировании часто используется увеличение или уменьшение значения переменной на единицу, а затем присваивание ей этого значения **x=x+1** или **x=x-1.** Сокращенная запись этих действий в JavaScript**:** **x++** (инкремент) и **x--(**декремент). Такой же результат будет получен при использовании выражения **++x** и **--x**. В первом случае использовалась постфиксная, а во втором - префиксная форма записи.

В примере 2 демонстрируется использование префиксной и постфиксной форм операции инкремента и декремента.

Пример 2

```
<html>
<script  language="JavaScript" type="text/javascript">
var i=0;

var result=0;
//префиксная форма инкремента
result=++i;
document.write("префиксная форма инкремента  ");
document.write(result+"<br>");
//сброс значения переменной i
i=0;
//постфиксная форма инкремента
result=i++;
document.write("постфиксная форма инкремента  ");
document.write(result+"<br>");
//сброс значения переменной i
i=0;
//префиксная форма декремента
result=--i;
document.write("префиксная форма декремента  ");
document.write(result+"<br>");
//сброс значения переменной i
i=0;
//постфиксная форма декремента
result=i--;
document.write("постфиксная форма декремента  ");
document.write(result+"<br>");
</script>
</html>
```

Результат работы примера 2 приведен на рис. 5.1

В первом случае переменной **result** присваивается значение **i+1**. Во втором случае результат равен значению переменной **i** до выполнения

инкремента **i**. В двух следующих примерах сначала переменной **result** присваивается значение декремента **i**, т.е. **i-1**, а затем значение, установленное до выполнения декремента.

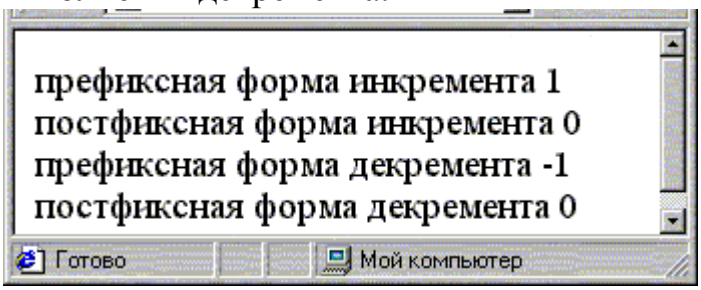


Рис. 5.1

Операции сравнения

Операции сравнения используются для *сравнения* выражений. Два операнда, соединенные операцией сравнения, образуют выражение сравнения. Результатом выражения сравнения являются значения **true** или **false.** В качестве операндов могут выступать числа, строки, переменные, свойства объектов или результаты работы функций. Рекомендуется сопоставлять выражения, относящиеся к одному и тому же типу. В противном случае JavaScript будет пытаться перевести данные из одного типа в другой, что не всегда приводит к корректным результатам. Операции сравнения приведены в таблице 5.3.

Таблица 5.3

Операция	Название	Пример	Результат
= =	равно	2= =4	false
!=	не равно	2!=4	true
>	больше	2>4	false
<	меньше	2<4	true
>=	больше либо равно	2>=4	false
<=	меньше либо равно	2<=4	true

Строковые операции

Строковые операции включают в себя операции сравнения и операцию конкатенации (объединения) строк, которая обозначается знаком **+.** Например, "**Java"+"Script".** Результатом будет строка "**JavaScript**".

Если хотя бы один из операндов является строкой, то результат объединения данных также будет строкой. В случае, когда первый операнд строка, все остальные операнды будут преобразованы в строки. Например, "33"+44. Результатом будет строка "3344". Если строкой является не первый операнд, то вначале выполняется сложение чисел, а

затем объединение со строковым операндом. Например, 44+5+"33". Результатом будет строка "4933".

Условная операция

Условная операция имеет следующий синтаксис:

Выражение сравнения ? Значение_если_истина: Значение_если_ложь

Если выражение сравнения имеет значение **true**, то возвращается **Значение_если_истина**, в противном случае, если выражение сравнения имеет значение **false**, возвращается **Значение_если_ложь.** Операция сравнения позволяет присвоить полученное значение переменной.

В примере 3 производится вывод информации о том, является ли високосным указанный год. Будем считать високосным год, без остатка делящийся на 4. Переменной **current_year** присваивается значение года, введенное пользователем, а затем с помощью условной операции переменной **result** присваивается значение, зависящее от результата проверки условия **current_year%4= =0.** При вводе значения 2016, в окне браузера будет получен результат "**високосный**".

Пример 3

```
<html>
<script  language="JavaScript" type="text/javascript">
var current_year=prompt("Введите год");

var result=current_year%4= =0? "високосный":"не високосный";
document.write(result);
</script>

</html>
```

Операция typeof

Операция **typeof** возвращает тип данных, хранящихся в операнде. В таблице 5.4 приведены результаты действия этой операции над различными операндами.

Таблица 5.4

Тип данных	**Результат**
Неопределенный	Undefined
Неопределенный	Object
Булево выражение	Boolean
Число	Number
Строка	String
Объект	Object

Объект (функция)	Function

Булевы операции

Булевы или логические операции служат для построения логических выражений, в которых операндами являются выражения сравнения. Логические операции перечислены в таблице 5.5.

Таблица 5.5

Оператор	**Название**	**Синтаксис**	**Возвращаемое значение**
&&	AND (И)	Опер1&&Опер2	**true**, если оба операнда равны true, в противном случае **false**
||	OR (ИЛИ)	Опер1||Опер2	**true**, если хотя бы один операнд равен **true**, в противном случае **false**
!	NOT (НЕ)	!Опер	**true**, если операнд равен **false**, в противном случае **false**

Приоритет операций

Порядок, в котором JavaScript вычисляет значения различных частей выражения, зависит от используемых операций. Операции в порядке убывания приоритетов приведены в таблице 5.6

Таблица 5.6

Операции	**Названия**
++, --, -, !	Инкремент, декремент, отрицание, не
***, /, %**	Умножение, деление, целочисленное деление
+, -,+	Сложение, вычитание, конкатенация
<, >, <=, >=	Операции сравнения
= =,!=	Операции сравнения
&&, ||	И, ИЛИ
?:	Условная операция

=, +=, -=, *=, /=, %=	Присваивание и арифметическое присваивание

5.2.3. Массивы

Массивы позволяют сохранять несколько значений в одной переменной. Обычно эти значения связаны по какому-либо признаку. Каждое значение представляет собой *элемент* массива. Элементы массива различаются по их индексам. Обращение к элементу массива осуществляется по его имени и индексу, заключенному в квадратные скобки. Нумерация индексов начинается с *нуля*.

Для работы с массивом нужно выполнить его *объявление*, *заполнить* массив данными (или инициализировать его), и выполнить *обработку* элементов в зависимости от требуемого алгоритма.

Для объявления массива используется ключевое слово **var**, за которым следует имя массива. Так как в **JavaScript** массив является объектом (**Array**), для создания нового объекта используется ключевое слово **new**. Например, для объявления массива **myArray** нужно воспользоваться конструкцией:

```
var myArray=new Array();
```

Оператор **Array**() называется конструктором, т.к. позволяет сконструировать объект в памяти компьютера.

Чтобы объявить количество элементов в массиве, можно в объявлении массива указать соответствующее число в скобках после слова **Array**:

```
var myArray=new Array(5);
```

JavaScript позволяет добавлять и удалять элементы массива.

Инициализировать массив можно при его объявлении, или используя операцию присваивания. В первом случае значения элементов массива записываются в скобках после ключевого слова **Array** и отделяются друг от друга запятыми:

```
var myArray=new Array("Bob","Pit","Tom","Sam","John");
```

Во втором случае инициализация будет выглядеть следующим образом:

```
var myArray=new Array();
myArray[0]="Bob";
myArray[1]="Pit";
myArray[2]="Tom";
myArray[3]="Sam";
myArray[4]="John";
```

Простой способ инициализации массива включает в себя объявление переменной и задание значений элементам массива:

```
var days=["понедельник","среда","пятница","воскресенье"];
```

Количество элементов в указанном массиве определяется с помощью свойства ***length*** объекта **Array**. Чтобы выяснить, сколько элементов содержится в массиве **myArray** нужно воспользоваться выражением **myArray.length**.

В примере 4 приведен код, выводящий на экран дисплея значения всех элементов массива, второго элемента массива, и длину массива. Результат, полученный в браузере, приведен на рис. 5.2.

Пример 4

```
<html>
<script  language="JavaScript" type="text/javascript">
<!--
var myArray=new Array("Bob","Pit","Tom","Sam","John");
document.write(myArray+"<br>");
document.write("второй элемент массива="+myArray[1]+"<br>");
document.write("Длина массива="+myArray.length);
//-->
</script>
</html>
```

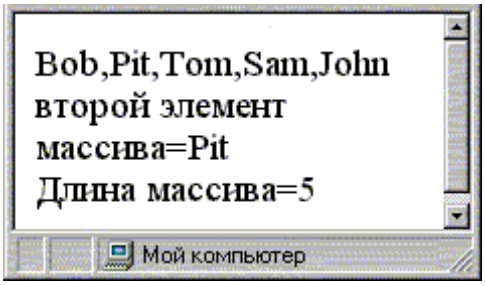


Рис. 5.2

В примере 5 к массиву **myArray** применен метод **sort()**, позволяющий упорядочить элементы массива (числовые данные - по возрастанию, а текстовые – по алфавиту). Результат, полученный в браузере, приведен на рис. 5.3

Пример 5

```
<html>
<script  language="JavaScript" type="text/javascript">
var myArray=new Array("Bob","Pit","Tom","Sam","John");
document.write("Неупорядоченный массив="+myArray+"<br>");
document.write("Упорядоченный массив="+myArray.sort());
</script>
</html>
```

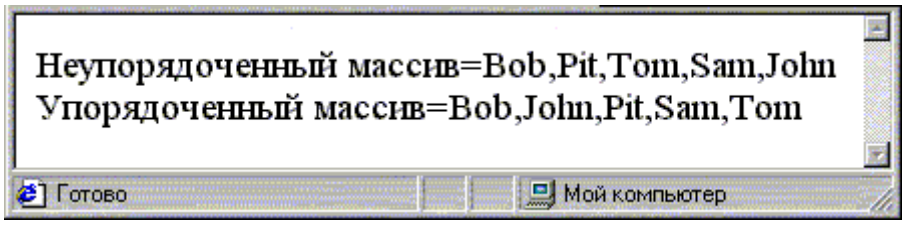


Рис. 5.3

Добавление элемента массива *перед первым* происходит с помощью метода **unshift()**, а добавление в конец массива – методом **push()**.

Конструкции

```
myArray.unshft("Коля");
myArray.push("Вася");
document.write("Новый массив="+myArray);
```

выведут на экран

Новый массив=Коля,Bob, Pit, Tom, Sam, John,Вася

Удаление элементов происходит с помощью метода **pop()** – последнего, а метода **shift()** – первого.

Метод **concat**() позволяет объединить массивы в новый массив. Следующая конструкция иллюстрирует действие этого метода.

```
var color1=["red","orange","yellow"];
var color2=["green","lightblue","blue","violet"];
var spectr=color1.concat(color2);
document.write("spectr="+spectr);
```

Результат, полученный на экране будет иметь вид:

spectr=red,orange,yellow,green,lightblue,blue,violet

JavaScript не поддерживает многомерные массивы, а оперирует только с одномерными. Однако он позволяет имитировать многомерные массивы, создавая один массив внутри другого. В примере 6 приведен двумерный массив **names_array**, который состоит из трех женских и трех мужских имен. Вывод элементов массива выполнен сначала построчно (в каждой строке – элемент массива), а затем с помощью двойного цикла (см. п. 5.2.5) женские имена записаны в первой строке, а мужские – во второй.

Пример 6

```
<html>
<head>
<body>
<script   type="text/javascript">
var names_girls=[" Ann "," Kate "," Mary "];
var names_boys=[" Bob "," Tom "," John "];
var names_array=[ names_girls, names_boys];

//вывод элементов массива построчно
```

```
document.write("names_array[0][0]="+ names_array[0][0]+"<br>");
document.write("names_array[0][1]="+ names_array[0][1]+"<br>");
document.write("names_array[0][2]="+ names_array[0][2]+"<br>");
document.write("names_array[1][0]="+ names_array[1][0] +"<br>");
document.write("names_array[1][1]="+ names_array[1][1]+"<br>");
document.write("names_array[1][2]="+ names_array[1][2]+"<br>");
//вывод с помощью цикла (i-номер строки, j-номер столбца)
for (i=0;i<2;i++){
  for (j=0;j<3;j++){
document.write(names_array[i][j]+" ");
}
document.write("<br>");
}
</script>
</body>
</html>
```

5.2.4. Использование операторов в языке JavaScript

Операторы в языке JavaScript обычно записываются в одной строке и заканчиваются точкой с запятой. Если разделители не установлены, то JavaScript устанавливает их самостоятельно.

5.2.4.1. Оператор variable

Оператор *variable* позволяет объявить компьютеру о существовании переменной и обеспечить отведение части памяти под эту переменную. Например,

```
var my_var;
```

Для объявления нескольких переменных нужно отделить их друг от друга запятой, например,

```
var my_var,  my_var1, my_var2;
```

5.2.4.2. Операторы принятия решений if и switch

Оператор **if** позволяет выполнить одну из двух альтернативных групп операторов в зависимости от значения проверяемого выражения (условия), которое может быть истинным (**true**) или ложным (**false**). Оператор **if** может не содержать часть, обеспечивающую обработку значения **false**.

В JavaScript следующие значения по определению равны **false**: *ноль*, *пустая строка* (""), **null** и **undefined** (значение неопределенной переменной).

Оператор **if** имеет следующий синтаксис:

```
if (выражение) {
```

```
true операторы;
}
else {
false операторы;
}
```

Пример 7 использует исходные данные примера 4. В нем производится вывод информации о том, является ли високосным указанный год. Будем считать високосным год, без остатка делящийся на 4. Переменной **current_year** присваивается значение года, введенное пользователем, а затем в зависимости от результата проверки условия **current_year%4= =0** на экран выводится соответствующее сообщение. Если значение условия равно **true** , то выводится сообщение "**високосный",** при значении **false** выводится сообщение " **не високосный".** При вводе значения 2016, в окне браузера будет получен результат "**високосный**".

Пример 7

```
<html>
<script  language="JavaScript" type="text/javascript">
var current_year=prompt("Введите год");
if (current_year%4==0) {
document.write("високосный");
}
else {
document.write("не високосный");
}
</script>
</html>
```

Если для принятия решения нужно поверить несколько условий, то можно использовать вложенные операторы **if**. Для формирования составных условий применяются логические операции **И, ИЛИ** и **НЕ**.

Вложенный оператор **if** имеет следующий синтаксис:

```
if (выражение1) {
true операторы1;
}
else  if (выражение2){
true операторы2;
}
else {
false операторы;
}
```

Данная конструкция работает следующим образом. Сначала проверяется **выражение1.** Если результат **true**, то выполняется группа операторов **true операторы1.** Все остальные операторы игнорируются. Если результат **false**, проверяется **выражение2.** Если результат **true**, то выполняется группа операторов **true операторы2.** Все остальные операторы игнорируются. В противном случае выполняются **false операторы.**

В примере 8 рассматривается работа вложенного оператора **if** , а также использование булевых операций **И** и **ИЛИ**. Пользователь вводит значение своего возраста, которое сохраняется в переменной **my_age**. Код JavaScript выдает сообщение о возрастной группе, к которой принадлежит пользователь. Возрастные группы организованы с шагом в 20 лет. Их формирование обеспечивается использованием логической операции **И (&&),** например, (**my_age <=40 && my_age >20)**. Кроме того существует группа, куда входят лица младше 20 и старше 80 лет. Она формируется с помощью логической операции **ИЛИ (||)** - **((my_age <=20) || (my_age >80)).**

Пример 8

```
<html>
<script  language="JavaScript" type="text/javascript">
var my_age=prompt("Введите ваш возраст");
if ((my_age <=20) || (my_age >80)) {
document.write("ваш возраст меньше 20 или больше 80");
}
else  if (my_age <=40 && my_age >20) {
document.write("ваш возраст от 20 до 40");
}
else  if (my_age <=60 && my_age >40) {
document.write("ваш возраст от 40 до 60");
}
else  {
document.write("ваш возраст от 60 до 80");
}
</script>
</html>
```

После ввода значения возраста в переменную **my_age** проверяется первое условие **((my_age <=20) || (my_age >80)).** Если оно равно **true** , выводится сообщение о данной возрастной группе. При значении **false** проверяется условие, находящееся в части **else if.** Если оно равно **true** выводится сообщение о следующей возрастной группе, в противном случае проверяется условие, находящееся в следующей части **else if.** Если оно

равно **true** выводится сообщение о соответствующей возрастной группе. Если ни в одной части значение **true** не возвращается, выполняется оператор, стоящий после ключевого слова **else.**

5.2.4.3. Оператор switch

При необходимости проводить большое число проверок, удобно использовать оператор **switch.** Оператор **switch** имеет следующий синтаксис:

```
switch (выражение){
case  выражение1 :
        операторы1;
        break;
case  выражение2 :
        операторы2;
        break;
        .
        .
        .
default:
        операторы;
        break;
        }
```

Сначала вычисляется **выражение**, которое может быть числом, строкой или булевым значением **true** или **false**. **Выражение1**, **выражение2 и т.д.** являются возможными значениями **выражения. Операторы1, операторы2 и т. д. –** это те операторы, которые выполняются в случае совпадения значений **выражения и** соответствующего **выражениян**. Если такого совпадения не произошло, то выполняются операторы, стоящие в части **default (**по умолчанию)**.** Чтобы закончить выполнение оператора **switch** после отработки одного из случаев совпадения выражений, каждая часть **case** заканчивается оператором **break.**

В примере 9 выводится *название* дня недели в зависимости от его порядкового номера, которое ввел пользователь. Единице соответствует "понедельник", двойке – "вторник" и т.д. до семи. Если пользователь введет число, не входящее в данный диапазон, то получит сообщение о невозможности нахождения соответствия между названием дня недели и и его порядковым номером.

Пример 9

```
<html>
<script  language="JavaScript" type="text/javascript">
var day_number=prompt("Введите номер дня недели");
```

```
switch (day_number){
case  "1" :
        document.write("понедельник");
        break;
case  "2" :
        document.write("вторник ");
        break;
case  "3" :
        document.write("среда ");
        break;
.
.
.
        document.write("воскресенье ");
        break;
default:
        document.write("не верно введен номер ");
        break;
        }
</script>
</html>
```

5.2.5. Операторы организации циклов

5.2.5.1. Операторы for и while

В языке JavaScript циклическое повторение блока кода при *истинности* условия осуществляется с помощью операторов **for** и **while.**

*Оператор **for***

Оператор **for** позволяет повторить блок кода *заданное* количество раз. Синтаксис этого оператора следующий:

```
for  ( var переменная цикла=начальному значению; выражение; шаг изменения переменной цикла)
{
  операторы;
}
```

Первый элемент в круглых скобках *инициализирует переменную цикла* т.е. присваивает ей начальное значение. *Переменная цикла* ведет подсчет количества повторений цикла.

Выражение содержит проверяемое условие. *Операторы* цикла будут повторяться до тех пор, пока условие имеет значение **true** . После каждого прохождения цикла условие проверяется вновь.

В последней части происходит изменение значения переменной цикла. Наиболее часто используемой операцией является инкрементирование, т.е. увеличение переменной цикла на единицу. Однако шаг изменения переменной цикла может быть выбран любым.

В фигурных скобках находится часть кода, которая повторяется. Если *выражение* изначально равно **false**, то код *операторов* игнорируется.

В примере 10 оператор цикла использован для заполнения значениями имен массива, состоящего из пяти элементов. Переменная цикла **counter** сначала имеет значение 0, затем проверяется *выражение*, которое имеет значение **true.** После выполнения кода, заключенного в фигурные скобки, выполняется операция инкрементирования и вновь проверяется значение выражения. Если оно по-прежнему равно **true** , происходит повторное выполнение цикла. Так продолжается до тех пор, пока выражение не будет иметь значение **false**. Тогда цикл закончится, и будут выполнены операторы, стоящие за циклом.

Вывод на экран значений элементов массива происходит после ввода имени с помощью инструкции **alert** внутри цикла**,** а также всех значений массива после завершения цикла.

Пример 10

```
<html>
<script  language="JavaScript" type="text/javascript">
var myArray=new Array();
for (counter=0; counter<=4;counter++)
{
  myArray[counter]=prompt("Input name");
  alert(myArray[counter]);
}
 document.write(myArray);
</script>
</html>
```

Если необходимо прервать выполнение цикла при достижении каких-либо условий, используют оператор **break.** В случае же необходимости пропустить несколько команд и вновь начать прохождение цикла используют оператор **continue**.

Например, прекратить ввод значений элементов массива при появлении имени "Маша". В этом случае код примера 10 будет выглядеть следующим образом.

Пример 11

```
< <html>
<script  language="JavaScript" type="text/javascript">
var myArray=new Array();
for (counter=0; counter<=4;counter++)
{
  myArray[counter]=prompt("Input name");
  if (myArray[counter] = ="Маша")
{
break
}
}
 document.write(myArray);
</script>
</html>
```

В примере 6 приведен двумерный массив **names_array**, который состоит из трех женских и трех мужских имен. Вывод элементов массива выполнен с помощью двойного цикла **for.** При этом женские имена записаны в первой строке, а мужские – во второй.

```
//вывод с помощью цикла (i-номер строки, j-номер столбца)
for (i=0;i<2;i++){
  for (j=0;j<3;j++){
document.write(names_array[i][j]+" ");
}
document.write("<br>");
}
```

Оператор **for…in**

Оператор **for…in** позволяет организовать цикл, используя все элементы массива, не зная, сколько их на самом деле. Например, инициализируем массив, состоящий из трех значений **var myArray=new Array("Pit","Bob","Tom");**

Для обращения к элементам массива с помощью оператора **for…in,** нужно записать следующий код:

```
var counter;
for (counter in myArray)
{
  document.write(myArray[counter]);
}
```

*Оператор **while***

Оператор **while** позволяет многократно выполнить цикл, пока проверяемое условие остается истинным. Этот оператор используется в тех случаях, когда число повторений цикла заранее не известно. Оператор **while** имеет следующий синтаксис:

```
while (выражение)
{
  операторы;
}
```

Операторы будут выполняться до тех пор, пока значение *выражения* остается истинным. Если *выражение* имеет значение **false** с самого начала, то цикл не выполнится ни разу.

В примере 12 пользователю предлагается ввести в диалоговое окно свой возраст. Если будет введена нечисловая информация, функция **isNaN** возвратит значение **true,** и цикл будет продолжен до тех пор, пока в диалоговое окно не будет введено число.

Пример 12

```
<html>
<script  language="JavaScript" type="text/javascript">
var user_age;
while (isNaN(user_age)==true)
{
user_age=parseInt(prompt("Пожалуйста, укажите свой возраст",""))
}
document.write("Ваш возраст= "+user_age)
</script>
</html>
```

*Оператор **do…while***

Оператор **do…while** имеет следующий синтаксис:

```
do
{
операторы;
}
while (выражение)
```

Оператор **do**…**while** является оператором с постусловием. Это означает, что цикл выполняется хотя бы один раз, независимо от значения *выражения***.** Продолжение цикла происходит при значении *выражения* - **true**.

В примере 13 применен цикл **do**…**while** для решения задачи из примера 12.

Пример 13

```
<html>
<script  language="JavaScript" type="text/javascript">
var user_age;
do
{
user_age=parseInt(prompt("Пожалуйста, укажите свой возраст",""))
}
while (isNaN(user_age)= =true)
document.write("Ваш возраст= "+user_age);
</script>
</html>
```

*Оператор **continue***

Оператор **continue (**продолжить**)** применяется в сочетании с операторами **for, for…in, while, do…while.** Этот оператор заставляет JavaScript перейти к выполнению следующего цикла и не выполнять операторы, стоящие за *continue***.** В примере 14 из 10 чисел натурального ряда, обозначенных идентификатором **x**, на экран будут выведены только нечетные числа. Для этого используется условие **x%2= =0**.

Пример 14

```
<html>
<script  language="JavaScript" type="text/javascript">
var x=0;
while (x<10)
{
x++;
if (x%2= =0)
{
continue
}
alert(x);
}
</script>
</html>
```

5.2.6. Создание сценариев с помощью функций и событий

5.2.6.1. Функции

При решении конкретных задач, стоящих перед разработчиками web-страниц, зачастую необходимо разбить сценарий на части, каждая из которых служила бы определенной цели. Например, для связи с каким-то браузером сначала нужно узнать его версию и возможности. Перед отправкой формы на сервер желательно проверить корректность ее заполнения. Если страница предназначена для учета проданного товара, то можно подсчитать стоимость любой комбинации этих товаров, а для отдела маркетинга иметь сведения о проданных товарах.

JavaScript предоставляет возможность разбиения сценария на части, известные как *функции*. Функция – это сценарий, отделенный от остальной программы и имеющий имя. Для предотвращения ошибок желательно размещать функцию в разделе **<head>.**

Синтаксис функций JavaScript следующий:

```
function имяФункции ([аргумент1][,…,аргументN]) {
  операторы;
}
```

Аргументы (или параметры) используются для вычислений. Они указываются в скобках и разделяются запятыми. Аргументы являются необязательными, однако скобки после имени функции нужны в любом случае.

Операторы функции заключаются в фигурные скобки.

Вызов функции

Функция выполняется в момент ее *вызова*. Вызов функции происходит по имени, при этом можно передать функции необходимые аргументы. Вызов функции может осуществляться *либо* путем включения в сценарий оператора, содержащего имя функции и необходимые аргументы, *либо* в момент наступления *события*, ее активизирующего.

Примером первого варианта может служить функция выдачи приветствия пользователю, не содержащая параметров. Вызов производится по имени функции, за которым стоят пустые круглые скобки.

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
function greeting(){
  var user_name=prompt("Enter your name");
  document.write("Hi, "+ user_name+"!");
}
greeting();
</script>
</head>
</html
```

Эта же функция, содержащая параметр, в качестве которого используется имя, введенное пользователем, приведена ниже.

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
function greeting(user_name){
  document.write("Hi, "+ user_name+”!”);
}
var name;
name=prompt("Enter your name");
greeting(name);
</script>
</head>
</html>
```

Локальные и глобальные переменные

В последнем примере переменная **name** объявлена *вне* функции – она является *глобальной* переменной и доступна в любом месте сценария. Переменные же, объявленные внутри функции *видны* только в ней. Код вне функции не имеет к ним доступа – это *локальные* переменные. Так, переменная **user_name** из предыдущего примера, объявленная внутри функции является *локальной* переменной и не доступна вне функции. Параметры функции также не видны за ее пределами.

Внутри функции нельзя объявлять переменные с именами, совпадающими с именами параметров функции. Локальные переменные и параметры функции имеют *время жизни*, которое заканчивается после выполнения кода функции. Чтобы сохранить и передать значение какой-либо переменной в сценарий, используется оператор **return.** Примеры его применения будут приведены далее.

События и обработчики событий

JavaScript является языком, управляемым *событиями*. События возникают при некоторых действиях пользователя или программно, т. е. из-за включения в сценарий соответствующего кода. Загрузка новой страницы в браузере, щелчок мыши, перемещение ее указателя – все это является событиями, которые используются для активизации различных функций. Наиболее часто используемыми событиями являются событие **Load**, возникающее при загрузке страницы, событие **Click**, возникающее при щелчке мыши, **Dblclick**-при двойном щелчке мыши, а также события **Mouseover** и **Mouseout**, возникающие при наведении и сведении мыши с объекта.

Реакция сценария на событие происходит с помощью *обработчика события***.** Обработчиком события чаще всего является функция JavaScript**.** Обработчик события может быть подключен с помощью атрибута события, который размещен в объекте, в котором это событие произошло и формируется добавлением префикса *on* к названию события. Например, событию **Click** соответствует атрибут **onClick,** а событию **Load –** атрибут **onLoad.** Такой атрибут добавляется к соответствующему дескриптору (объекту) **HTML.** Например, **<body onLoad>.**

В простейшем случае, если при наступлении события нужно выполнить один оператор, то обработчик события может представлять собой инструкции JavaScript, не образующие функцию.

В примере 15 а приветствие пользователю будет выдаваться при щелчке мыши (событии **Click)** на кнопке. При этом подключается обработчик события (функция **greeting**).

Пример 15 а

```
<html>
<head>
</head>
<body>
<form>
<input  type="button" value="greeting" id="but1"  onclick="greeting()">
</form>
<script  language="JavaScript" type="text/javascript">
function greeting(){
  var user_name=prompt("Enter your name");
  document.write("Hi, "+ user_name);
}
</script>

</body>
</html>
```

В примере 15 б используется другой способ вызова обработчика события **Click**. Сохраним обращение к кнопке в переменной **bb** и в ответ на событие **Click** на кнопке вызовем функцию **greeting.**

Пример 15 б

```
<html>
<head>
</head>
<body>
<form>
```

```
<input  type="button" value="greeting" id="but1" >
</form>
<script  language="JavaScript" type="text/javascript">
function greeting(){
  var user_name=prompt("Enter your name");
  document.write("Hi, "+ user_name);
}
var bb=document.getElementById("but1");
bb.onclick= greeting;
</script>

</body>
</html>
```

Возвращаемое значение функции

В предыдущих примерах параметры функции использовались для передачи значений в функцию. Для возврата результата обратно в сценарий используется оператор *return*. Следует помнить, что *передавать* функции можно *несколько* параметров, а *возвращать* с помощью оператора *return* функция может только *одно* значение.

В примере 16 производится перевод температуры по шкале Цельсия в температуру по шкале Фаренгейта. Формула пересчета: $t_F=1.8* t_C+32$.

Функция пересчета **Func_calc** принимает значение температуры по Цельсию как параметр и возвращает преобразованное значение (результат работы функции) с помощью оператора *return*. Результат присваивается переменной **ans_Fah**.

Пример 16

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
function Func_calc(cels) {
  var ans=(1.8*Number(cels)+32)
  return(ans)
}
var c=prompt("Введите температуру в градусах по Цельсию"," ");
ans_Fah=Func_calc(c);
document.write(c+" градусов по Цельсию равно "+ans_Fah+" градусам
по Фаренгейту ");
</script>
</head>
</html
```

Модифицируем пример 16 таким образом, чтобы ввод исходной температуры и преобразование температуры осуществлялось разными функциями. Функция преобразования **Func_calc** будет вызвана из функции ввода **Input_cels**, а запуск функции ввода будет происходить при загрузке страницы, которая сопровождается событием **Load**. Обработчик события связан с фукцией **Input_cels** и выглядит следующим образом: **onLoad=”Input_cels()”**

Пример 17

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
function Input_cels(){
      var cels=prompt("Введите температуру в градусах по Цельсию","
");
      ans_Fah=Func_calc(cels);
      document.write(cels+" градусов по Цельсию равно "+ans_Fah+"
градусам по Фаренгейту ");
}
function Func_calc(cels) {
         var ans=(1.8*Number(cels)+32)
        return(ans);
}
</script>
</head>
<body onLoad=”Input_cels()”>
</body>
</html>
```

5.2.7. Краткое введение в объекты

5.2.7.1. Объекты JavaScript

Программируемые на языке JavaScript элементы, обладающие набором свойств и методов – это объекты. JavaScript содержит ряд *встроенных* объектов, которые расширяют возможности программного кода, и позволяют решать разнообразные задачи.

Каждый из объектов JavaScript имеет набор свойств и методов, позволяющих манипулировать данными определенного типа. Например, объект **Array** содержит методы для манипуляции массивами и свойствами для извлечения из них информации. В большинстве случаев, прежде чем воспользоваться методами и свойствами объекта, его нужно предварительно создать. Например,

var myArray=new Array();

В левой части объявляется переменная по имени **myArray**. Ключевое слово **new** предписывает интерпретатору создать новый объект. Далее следует конструктор объекта **Array()**. Подобные конструкторы есть у большинства объектов. Особенностью объектов является тот факт, что переменные объектов содержат не данные, а **ссылки** на фрагменты памяти, в которых находятся данные.

Доступ к свойствам объекта достаточно прост. Например, к свойству **length** массива **myArray** можно обратиться: **myArray.length.** Некоторые свойства объектов можно изменять.

Методы могут быть вызваны так же как свойства: **myArray.sort().** Данный метод отсортирует массив **myArray.** Подобно функциям, некоторые объекты принимают параметры, указываемые в скобках после имени объекта. Скобки ставятся независимо от наличия параметров.

*Объект **String***

Объект позволяет изменять и форматировать текстовые строки, а также выделять внутри них части строки. Строка представляет собой последовательность символов, причем первый символ строки находится на нулевой позиции.

Свойство **length** позволяет определять длину строки, поэтому код:

var string1="JavaScript";

document.write(string1.length);

выведет на экран число 10.

Методы **substr**() и **substring**() позволяют выделять часть строки. Новая строка может быть сохранена в какой-либо переменной для использования в выражении. Оба метода возвращают подстроку, но требуют различные параметры.

Метод **substr()** принимает два параметра, первый из которых указывает позицию первого символа вырезаемой подстроки, а второй – длину подстроки.

Метод **substring()** принимает два параметра, первый из которых указывает начальную позицию вырезаемой подстроки, а необязательный второй – конечную (не входящую в строку). Если его нет, то вырезается подстрока от начальной позиции до конца строки.

Например, из строки **JavaScript** нужно вырезать строку **Java.**

Используя метод **substr():**

var string1="JavaScript";

var string2=string1.substr(0,4);

alert(string2);

Используя метод **substring():**

```
var string1="JavaScript";
var string2=string1.substring(0,4);
alert(string2);
```

Методы **toLowerCase() и toUpperCase()** служат для перевода текста соответственно в верхний и нижний регистры. Эти методы удобно использовать для снятия чувствительности к регистру двух строк, подлежащих сравнению. Например, нужно сравнить строку **string1** и строку **string2,** не заботясь о регистре символов.

```
var string1="javaScript";
var string2="Javascript";
if (string1.toLowerCase()= =string2.toLowerCase()){
alert("OK");}
```

В результате работы приведенного кода на экране в окне предупреждения будет выведено OK.

*Объект **Math***

Объект Math обеспечивает использование в языке JavaScript математических функций и констант. Свойства этого объекта позволяют получить число π, логарифмы 2 и 10, а также числа E с различными основаниями, постоянную Эйлера. Методы позволяют проводить такие математические операции, как извлечение квадратного корня, возведение в степень, округление, тригонометрические вычисления, вычисление логарифма, определение минимального и максимального из двух чисел, а также генерацию случайных чисел в интервале от 0 до 1.

В примере 18 приведено округление до произвольного числа десятичных знаков. JavaScript не содержит стандартную функцию, которая бы это делала. Сценарий содержит две функции: **inp** и **dec_round.** Первая функция **inp** обеспечивает ввод числа, подлежащего округлению (например, 12.543) и количества десятичных знаков (например, 2), а затем вызывает функцию, производящую округление **dec_round**.

В первой строке этой функции переменной **div** присваивается результат возведения числа 10 в степень, равную заданному количеству десятичных знаков (для нашего случая 10 возводится в квадрат (получается 100). В следующей строке десятичная точка в округляемом числе сдвигается на два разряда вправо (получаем 1254.3), затем число округляется до ближайшего целого (1254) и делится на переменную **div** (100). В результате получим 12.54. Этот результат возвращается вызывавшему коду и выводится на экран. Вызов функции **inp** происходит при загрузке страницы.

Пример 18

```
<html>
```

```
<head>
<script  language="JavaScript" type="text/javascript">
function dec_round(num,dec_places){
  var div=Math.pow(10,dec_places);
  num=Math.round(num*div)/div;
  return (num);
}
function inp(){
 num1= prompt("Введите число, подлежащее округлению"," ");
 num2= prompt("Введите число десятичных знаков"," ");
document.write(num1+", округленное до"+num2+" знаков:");
document.write(dec_round(num1,num2));
}
</script>
</head>
<body onLoad="inp()">
</body>
</html>
```

В примере 19 приведен сценарий, обеспечивающий генерацию 10 случайных чисел в диапазоне от 0 до 600. Стартовой точкой является использование метода **random** объекта **Math.**

Пример 19

```
<html>
<body >
<script  language="JavaScript" type="text/javascript">
  var count;
  var num_random;
for (count=0; count<10;count++)
{
num_random=Math.floor(Math.random()*600);
document.write(num_random+"<br>")
}
</script>
</body>
</html>
```

*Объект **Date***

Объект **Date** позволяет манипулировать датами и временем в программах, написанных на JavaScript. Время в JavaScript измеряется в

миллисекундах, т.е. в тысячных долях секунды: Отсчет производится с 1 января 1970 года. Даты до этого времени будут измеряться в миллисекундах со знаком минус.

С помощью объекта **Date** можно выяснять текущую дату и время, задавать произвольную дату и время, выполнять вычисления и преобразовывать даты в строки, а также узнать, появились ли на сайте изменения с момента его последнего посещения пользователем с помощью **cookie**.

*Создание объекта **Date***

Перед работой с датой или временем нужно создать копию объекта **Date**. Это достигается вызовом функции (конструктора) **Date**() с ключевым словом **new.** Например,

my_date=new Date();

При этом дата и время считываются с пользовательского компьютера и сохраняются в переменной **my_date.**

Создать и определить новый объект можно используя аргументы функции **Date**.

my_date=new Date(949278000000); в этом случае задано число миллисекунд, прошедших с 0 часов 1 января 1970 года. В таком виде **JavaScript** хранит даты.

my_date=new Date(2016,0,1); задан год, месяц и день.

my_date=new Date(2016,0,1,0,0,0); задан год, месяц и день, а также часы, минуты и секунды.

my_date=new Date("31 January 2016"); дата задана в виде текстовой строки.

my_date=new Date("January 31, 2016"); дата задана в виде текстовой строки.

Получение даты и времени

Для чтения даты и времени объекта **Date** используются методы, приведенные в таблице 5.7

Таблица 5.7

Метод	Возвращаемое значение
getDate()	День месяца (от1 до 31)
getDay()	День недели (0-воскресенье,…,6-суббота)
getMonth()	Месяц (0-январь,…,11-декабрь)
getFullYear()	Год в четырехзначном формате
getHours()	Час (от 0 до 23)
getMinutes()	Минуты (от 0 до 59)

getSeconds	Секунды (от 0 до 59)
getMilliseconds	Миллисекунды (от 0 до 999)
getTime()	Миллисекунды с 1 января 1970 GMT

В примере 20 методы объекта **Date** используются для вывода дня, месяца и года на экран. В начале кода объявляется и инициализируется массив, в котором хранятся названия месяцев, т.к. объект **Date** имеет метод только для выдачи номера месяца. Затем создается новый объект **Date** – **dateNow** и соответствующим переменным присваивается значение дня, месяца и года. Ответ должен быть выдан в виде:

“Сейчас 7-ой день месяца Январь 2016 года”.

Для формирования окончаний "-ый", "-ий" и "-ой" используется оператор **switch**, который изменяет окончания слов в зависимости от значения дня даты.

Пример 20

```
<html>
<body >
<script  language="JavaScript" type="text/javascript">
   var months=new
   Array("Январь","Февраль","Март","Апрель","Май","Июнь","Июль","
   Август","Сентябрь","Октябрь","Ноябрь","Декабрь");
    var dateNow=new Date();
    var yearNow=dateNow.getFullYear();
    var monthNow=months[dateNow.getMonth()];
    var dayNow=dateNow.getDate();
var daySuffix;
switch(dayNow)
{
case 3:
case 23:
    daySuffix="-ий";
    break;
case 2:
case 6:
```

```
case 7:
case 8:
case 22:
case 26:
case 27:
case 28:
    daySuffix="-ой"
    break;
default:
    daySuffix="-ый"
    break;
}
document.write("Сейчас "+ dayNow + daySuffix +" день");
document.write(" месяца "+ monthNow );
document.write(" "+yearNow +" года");
//-->
</script>
</body>
</html>
```

Установка даты

Выполнение арифметических операций над датами требует изменения значения объекта **Date**. Для этого используются методы, приведенные в таблице 5.8.

Таблица 5.8

Метод	Устанавливаемое значение
setDate()	День месяца (от1 до 31)
setMonth()	Месяц (0-январь,…,11-декабрь)
setFullYear()	Год в четырехзначном формате
setHours()	Час (от 0 до 23)
setMinutes()	Минуты (от 0 до 59)
setSeconds	Секунды (от 0 до 59)
setMilliseconds	Миллисекунды (от 0 до 999)
setTime()	Миллисекунды с 1 января 1970

	GMT

Например, необходимо вывести на экран дату, которая наступит через 28 дней после текущей. Код, приведенный в примере 21, решает эту задачу. Вывод даты осуществляется использованием конструкции **document.write(nowDate);** где **nowDate –**имя созданного объекта.

Пример 21

```
<html>
<body >
<script  language="JavaScript" type="text/javascript">
  var nowDate=new Date();
var current_day=nowDate.getDate();
var new_day=nowDate.setDate(current_day+28);
document.write(nowDate);или
document.write(nowDate.getMonth()+" "+nowDate.getDate()+"
"+nowDate.getFullYear());
</script>
</body>
</html>
```

В примере 22 приведена функция, обеспечивающая вычисление количества дней между двумя датами. В качестве аргументов используются даты, между которыми нужно определить количество дней. Они преобразуются в формат миллисекунд с помощью метода **getTime**. В функции используется константа, содержащая количество миллисекунд в одном дне. Разность дат, взятая по модулю, делится на константу и затем с помощью метода **round** округляется до целого. Пользователь вводит даты в окна приглашения, которые используются для создания объектов **Date**.

Пример 22

```
<html>
<body >
<script  language="JavaScript" type="text/javascript">
<!--
  function num_days(date1,date2){
        var const_day=1000*60*60*24;
        var date1_ms=date1.getTime();
        var date2_ms=date2.getTime();
        var differ=Math.abs(date1_ms-date2_ms);
        return Math.round(differ/const_day);
  }
var date_user1=prompt("Input date","August 26,2016");
var date_user2=prompt("Input date","August 26,2016");
date11=new Date(date_user1);
```

```
date22=new Date(date_user2);
document.write("Число дней между датами: "+num_days(date11,date22));
//-->
</script>
</body>
</html>
```

Таймеры на web-страницах

На web-страницах можно создавать таймеры двух типов. Один срабатывает лишь раз, по истечении заданного промежутка времени, другой срабатывает периодически через определенные промежутки времени.

Для установки одноразового таймера нужно воспользоваться методом объекта **window setTimeout**("**код JavaScript", задержка_миллисек**). Код JavaScript может быть функцией или отдельными операторами. Второй параметр определяет задержку в миллисекундах, по истечении которой выполняется этот код.

Метод возвращает целое число, являющееся уникальным идентификатором таймера. С помощью этого идентификатора можно обнулить таймер. Для этого используется метод объекта **window clearTimeout**(**идентификатор таймера**)**.**

Например,

```
timerID=setTimeout("alert('Hi')",5000);
clearTimeout(timerID);
```

В примере 23 показан код, обеспечивающий вывод на экран часов, которые располагаются в текстовом поле формы. Функция **ShowTime** активизируется событием **Load.** Функция формирует текущее время, используя методы **getHours, getMinutes и getSeconds** объекта **Date.** В примере используется одноразовый таймер, который сработает в первый раз через секунду и активизирует функцию **showtime.** Функция считает данные объекта **Date** (следующий момент времени) и вновь активизирует таймер. Цикл будет повторяться до тех пор, пока не щелкнуть на кнопке “сброс таймера”**,** которая запускает функцию сброса таймера **StopTimer().** Для продолжения работы таймера нужно щелкнуть на кнопке “старт таймера”, которая вновь запустит функцию **showtime**.

Пример 23

```
<html>
<head >
<title>
ЧАСЫ
</title>
```

```
<script language="JavaScript">
function showtime()
{
var now=new Date();
var hours=now.getHours();
var minutes=now.getMinutes();
var seconds=now.getSeconds();

if(minutes<10)
{minutes="0"+minutes};

if(seconds<10)
{seconds="0"+seconds};

document.clock.face.value=hours+":"+minutes+":"+seconds;
timerID=setTimeout("showtime()",1000);
}
function StopTimer(){
clearTimeout(timerID)
}
</script>
</head>
<body onLoad="showtime()">
<form name="clock">
<input name="face" type="text" size=7 value="...ini..">

<input name="face1" type="button" size=7 value="стоп таймер"
onClick="StopTimer()">

<input name="face1" type="button" size=7 value="старт таймер"
onClick="showtime()">
</form>
</body>
</html>
```

Результат, полученный в браузере, приведен на рис. 5.4.

Таймеры, срабатывающие через регулярные интервалы времени устанавливаются с помощью метода **setInterval**() и сбрасывается с помощью метода **clearInterval**(). Метод **setInterval**() имеет те же параметры, что и **setTimeout**(), только второй параметр определяет не просто задержку перед выполнением кода, а интервал в миллисекундах между повторными срабатываниями.

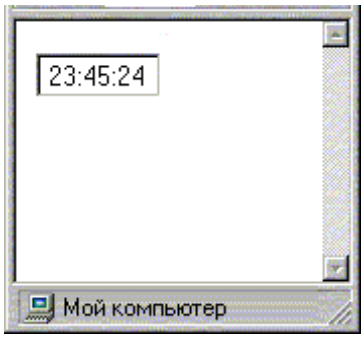


Рис. 5.4

Простая анимация

Использование таймеров позволяет создать простую анимацию на web-странице. Примеры 24_1 и 24_2 посвящены этой теме.

В примере 24_1 приведен код, позволяющий менять через каждые 3 секунды рекламные изображения. Для остановки процесса нужно щелкнуть на изображении и тем самым включить функцию сброса таймера **Stop**().

Имена изображений составлены следующим образом: **AdImage1.jpg, AdImage2.jpg, AdImage0.jpg.** Для получения последовательности чисел 1,2,0 используется счетчик по модулю 3 (операция %). Номера загруженных изображений хранятся в глобальной переменной **currentImgNumber.**

Для смены изображения эта переменная увеличивается на единицу, затем изменяется свойство **src** объекта **img** с помощью инструкции **document.imgAd.src="AdImage"+currentImgNumber+".jpg";**

Пример 24_1

```
<html>
<head >
<script language="JavaScript">
var currentImgNumber=0;
function switchImage()
{
document.imgAd.src="AdImage"+currentImgNumber+".jpg";
currentImgNumber++;currentImgNumber=currentImgNumber%3;
}
function start(){
timerID=setInterval("switchImage()",3000);
}
function stop(){
clearInterval(timerID);
}
</script>
</head>
```

```
<body onLoad="start()">
<img src="AdImage1.jpg" name="imgAd" onClick="stop()">
</body>
</html>
```

В примере 24_2 приведен программный код, отображающий перемещение по экрану параграфа, содержащего текст. Параграф позиционируется с помощью стилевых элементов, записанных в файле **my.css** и присоединяется к html-файлу в дескрипторе **link**. Файл с кодом **JavaScript my.js** присоединяется в дескрипторе **script**.

Для перемещения по горизонтали используется изменение позиции с помощью свойства **left** объекта **style**, которое изменяется от 70 до 400 px с шагом 5 px.

Пример 24_2

```
<html>
<head>
<link rel=stylesheet href="my.css"/>
</head>
<body >
<p id="para1">Движущийся текст</p>
<form>
<input type="button" id="but" value="start">
<input type="button" id="but1" value="stop">
</form>
<script type="text/javascript" src="my.js"></script>
</body>
</html>
```

Файл my.js

```
var leftMargin=70;
var rightMargin=400;
var currentPos=leftMargin;
var step=5;
function move_t(){
document.getElementById("para1").style.left=currentPos;
currentPos+=step;
if(currentPos>rightMargin||currentPos<leftMargin)step=-step;
}
function start(){
ii=setInterval("move_t()",100);
}
function stop(){
```

```
clearInterval(ii);
}
var bb=document.getElementById("but");
bb.onclick=start;
var bb1=document.getElementById("but1");
bb1.onclick=stop;
```

Файл my.css

```
#para1{
position:absolute;
left:10px;
top:10px;
border:2px solid red;
font-size:24pt;
background-color:#f0f0f0;
color:black;
}
```

5.2.7.2. Объект Window. Работа с окнами

Новое окно для загрузки в него файла можно открыть, используя атрибут **target** дескриптора <**a**>:

```
<a href=URL  target="_blanck">go to URL</a>
```

В случае повторного использования такой конструкции браузер откроет еще одно окно и т. д. Вместо встроенных параметров **_blanck, _self, _parent, _top** можно дать новому окну имя и впоследствии загружать новые страницы в это окно. Тогда дескриптор <**a**> будет следующим:

```
<a href=URL  target="MyWindow">go to URL</a>
```

JavaScript позволяет открыть новое окно с помощью метода **window.open**(). Новое окно может быть использовано как для загрузки в него файла, так и для создания нового документа "на лету". Можно также управлять параметрами нового окна, его размещением и передвижением по экрану, выводить сообщения в строке состояния.

Метод **window.open**() можно использовать вместо атрибута **target**. В этом случае дескриптор <**a**> будет следующим:

```
<a href="javascript:void(0)"
onClick="window.open('URL','MyWindow')">go to URL</a>
```

Атрибут **href** имеет значение **javascript:void(0)** и отменяет штатную реакцию дескриптора на щелчок мышью по ссылке. Обработчик события подключается с помощью атрибута **onClick,** который использует метод **window.open(). URL**-это адрес страницы, которая открывается в новом окне,

а **MyWindow -** имя нового окна. Это имя может быть использовано *для помещения файла в именованное окно*.

Метод **window.open()** возвращает *ссылку* на созданное окно, которая может быть сохранена и использована для *обращения к свойствам и методам окна*.

Основными методами являются: **close**(), закрывающий окно, **focus**(), позволяющий расположить окно поверх других, **moveTo**() и **moveBy**(), позволяющие перемещать окно, **resizeTo**() и **resizeBy**(), изменяющие размер окна.

Из свойств можно отметить свойство **status**, позволяющее обращаться к статусной строке.

Код сценария, позволяющий создать новое окно и *закрыть* его, приведен в примере 25. В переменной **new_window** сохранена *ссылка* на новое окно, которое открывается с помощью метода **window.open**(). Для закрытия нового окна нужно воспользоваться этой ссылкой и применить метод **close**() - **new_window.close**(). Стоит обратить внимание на то, что конструкция **window.close**() или **self.close**() закроет окно, содержащее данную конструкцию. Для закрытия другого окна следует использовать имя ссылки (переменной, в которой определен закрываемый объект).

Прежде чем метод **close**() будет использован, нужно проверить, существует ли объект, подлежащий закрытию, и не закрыт ли он уже. Если это имеет место, то на экран выводится соответствующее сообщение.

Пример 25

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
var new_window;
function open_window()
{
new_window=window.open("пример15.html","");
}
function close_window()
{
if(!new_window ||new_window.closed){
alert("Окно не открывалось или уже закрыто")}
else {
new_window.close();
}
}
</script>
</head>
```

```
<body>
<a href="javascript:void(0)" onClick="open_window()">Открыть новое
окно с примером 15 </a>
<br>
<a href="javascript:void(0)" onClick="close_window()">Закрыть окно с
примером 15</a>
</body>
</html>
```

При использовании метода **window.open()** для отображения окна с желаемыми функциональными возможностями нужно перечислить дополнительные параметры, обеспечивающие формирование этих возможностей через запятую так, как показано далее.

```
window.open("пример15.html","myWindow","width=200,height=100,le
ft=300,top=200,location,menubar,toolbar,resizable,status");
```

В примере 26 задаются размеры нового окна (200×200 пикселей) и его расположение по центру экрана.

Пример 26

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
var new_window;
function open_centered_window()
{
var win_height=200;
var win_width=200;
var win_left=(screen.width/2)-(win_width/2);
var win_top=(screen.height/2)-(win_height/2);
var win_Features="height=200,width=200";
win_Features=win_Features+",left="+win_left+",top="+win_top;
new_window=window.open("пример15.html","myWin",win_Features);
}
</script>
</head>
<body>
<a href="javascript:void(0)" onClick="open_centered_window()">Открыть
новое окно с примером 15 </a>
</body>
</html>
```

Для обращения к открывающему окну существует свойство **opener** объекта **window**. Например, конструкция, размещенная в новом окне:

window.opener.document.bgColor="red" изменит цвет фона открывшего окна на красный.

Вывод HTML кода в новое окно

Для создания html-кода в новом окне сначала создается новое окно с пустым первым параметром:

var newWindow=window.open(" ","myWindow", "width=150,height=150");

затем открывается документ, в который будет записан **HTML**-код:

newWindow.document.open();

теперь с помощью метода **document.write()** на страницу может быть выведен html-код:

newWindow.document.write("<h3> Hello!</h3>")

newWindow.document.write(" Welcome to my window ")

При вызове метода **write** текст будет добавляться к существующему уже тексту до тех пор, пока не будет вызван метод **close().**

Свойство **window.status** позволяет вывести сообщение в строке состояния. Использование таймеров дает возможность реализовать в строке состояния "бегущую строку". Код примера 27 демонстрирует одну из таких возможностей.

Пример 27

```
<html>
<head>
   <script  language="JavaScript" type="text/javascript">
i=0;
beg="Для просмотра текста доклада щелкните мышкой по заголовку!!!";
beg1=beg;
function bs(){
if (i<150){
window.status=beg1;
beg1=' '+beg1;
i++}
else
{beg1=beg;i=0;}
setTimeout('bs()',100)
}
</script>
</head>
<body  onload="bs()">
</body>
```

</html>

5.2.7.3. Объект Document. Работа с формами

Форма представляет собой интерактивный компонент web-страницы, предназначенный для заполнения пользователем. Информация, полученная от пользователя, может быть обработана непосредственно на web-странице, а может быть отправлена на web-сервер и сохранена в базе данных.

Форма содержит совокупность элементов управления: кнопок, флажков, текстовых полей, списков и т. п. Для создания формы на web-странице используется парный дескриптор **<form> </form>.** Этот дескриптор создает объект **Form**, используемый для доступа к форме. Атрибут **name** дает возможность ссылаться на форму. Все формы в документе образуют коллекцию или массив **forms**. Доступ к форме осуществляется следующим образом:

по имени, например, для формы **<form name="myForm"> </form>**

document.myForm или **. document.forms.myForm**

как к элементу массива **forms**,например, если созданная форма является первой на странице,

document.forms[0] или **document.forms["myForm"].**

Свойства и методы объекта Form

Для пересылки данных на сервер используется свойство **action.** Оно указывает **URL**, который будет обрабатывать данные формы. Свойство **method** устанавливает метод, с помощью которого данные формы будут отправлены на сервер и может принимать значения **Get** или **Post**.

Метод **submit**() позволяет отправить данные формы на сервер, а метод **reset**() служит для очистки формы и восстановления значений элементов управления по умолчанию.

Элементы формы

Элементы формы, которые называют также *элементами управления или полями* форм – это элементы, необходимые пользователю для ввода или управления данными.

Элементы формы являются объектами. Одним из способов доступа к этим объектам является использование массива элементов **elements[]** объекта **Form.** Ссылка на элемент может быть следующей:

myForm.elements[0]-для первого элемента формы с именем **myForm**

или **myForm.elements["myElement"],** где "**myElement**"-имя элемента,

или **myForm.myElement,.**

или **document.getElementById("id элемента"),** где используется **id** элемента**.**

Свойство **length** возвращает количество элементов в форме. Чтобы им воспользоваться, достаточно записать код **document.myForm.length**.

Элементы формы имеют некоторые общие свойства. Свойство **name** и **id** используются в сценарии для ссылки на элемент. Кроме того, когда данные формы пересылаются на сервер, вместе со значением элемента пересылается и его свойство **name**, чтобы было известно, к какому элементу относятся данные.

Большинство объектов имеет свойство **value**. Например, для текстового поля – это текст, введенный пользователем.

Для определения типа элемента существует свойство **type**.

Все элементы формы имеют методы **focus**() и **blur**(). Программно фокус устанавливается с помощью метода **focus**(). Метод **blur**() смещает фокус с объекта.

Кнопки

Самая распространенная группа элементов управления – кнопки. Можно использовать кнопки трех типов: **button**, **submit** и **reset** .

Для создания кнопки **button** используется дескриптор <**input**>. Например, для создания кнопки с надписью "**Click me**" и именем **myButton** нужно воспользоваться дескриптором:

```
<input type="button" name="myButton" value="Click me">
```

Событием, связанным с кнопкой, является событие **click**. В примере 32 приведен код, подсчитывающий количество щелчков по кнопке и отображающий это значение в надписи на кнопке.

Пример 32

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
var number_clicks=0;
function myButton_onclick()
{
number_clicks++
window.document.form1.myButton.value='Кнопка щелкнута '
+number_clicks+' раз';
}
</script>
<body>
<form name=form1>
<input type="button" name="myButton" value="Кнопка щелкнута 0
раз" onclick="myButton_onclick()">
</form>
```

```
</body>
</html>
```

Щелчок по кнопке **submit** порождает событие **click** объекта **Submit** или событие **submit** объекта **Form.**

Нажатие кнопки **reset** порождает событие **click** объекта **reset** и событие **reset** объекта **Form.** Пример использования кнопок **reset** и **submit** в сочетании с методами **reset**() и **submit**() объекта **Form** приведен ниже.

Пример 34

```
<html>
<head>
<script language="JavaScript" type="text/javascript">
function button_handler(){
var submit_ok=confirm("Щелкните на кнопке ОК для отправки данных"+"\n"+"Щелкните на кнопке CANCEL для восстановления значения по умолчанию")
if (submit_ok){
document.forms[0].submit()
}
else{
document.form1.reset()
}
}
</script>
<body>
<form name=form1>
Введите фамилию: <input type="text" name="text" value="фамилия">
<br><br>
<input type="button" value="submit / reset" onClick="button_handler()">
</form>
</body>
</html>
```

Результат работы кода представлен на рис. 5.8.

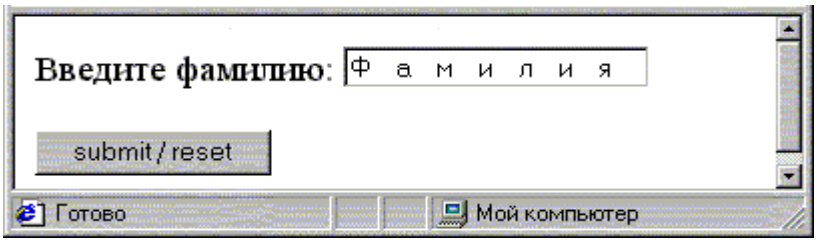


Рис. 5.8

Текстовые поля

Стандартное текстовое поле позволяет ввести строку текста. Для создания текстовых полей применяется конструкция **<input type="text">.** В результате создается объект **Text** со своим набором свойств, методов и событий. Атрибуты **value** и **placeholder** позволяют установить текст в элементе по умолчанию, а **size** и **maxlength** задают ширину поля в символах и максимальное количество символов, которые пользователь может ввести в поле. Кроме событий **focus**, **blur**, текстовые поля имеют часто используемое событие **change**, которое возникает, когда элемент теряет фокус и значение в текстовом поле отличается от того, что было до получения фокуса. Значение текстового поля может быть сохранено следующим выражением:

text_save=document.form1.text1.value, где **form1** - значение атрибута **name** формы, а **text1** - значение атрибута **name** текстового поля.

Текстовые поля, которые содержат строки, состоящие только из цифр, можно преобразовать в числовой тип данных, используя методы **parseInt**() и **parseFloat**().

Текстовые области

Текстовые области предназначены для ввода нескольких строк текста. Для создания текстовой области используется дескриптор **<textarea> содержание области</textarea>** . Дескриптор имеет специфические атрибуты **cols** и **rows,** которые определяют ширину области в символах и количество строк текста.

Объект **Textarea** имеет те же свойства, методы и события, что и объект **Text**. Объект **Textarea** имеет свойство **value.** Это свойство возвращает текст, заключенный между дескрипторами <**textarea>** и </**textarea>.**

Текстовое поле пароля

Текстовое поле пароля предназначено для ввода одной текстовой строки. Вместо символов строки в нем отображаются звездочки.

Объект **Password** создается с помощью дескриптора <**input**> , когда его атрибуту **type** присваивается значение **password.** Объект управляется аналогично объекту **Text.**

Скрытое текстовое поле

Скрытое текстовое поле может содержать текст и числа, не видные пользователю. Значения этого поля передаются на сервер как значения обычного текстового поля. Создание этого поля обеспечивается с помощью дескриптора <**input**>, когда его атрибуту **type** присваивается значение **hidden.** Объект управляется аналогично объекту **Text.**

Флажки

Для создания флажков применяется конструкция **<input type="checkbox">.** В результате создается объект **Checkbox** со своим набором свойств, методов и событий. Если флажок установлен, то свойство **checked** объекта **Checkbox** возвращает значение **true**, в противном случае - **false**. Для установки флажка по умолчанию в дескрипторе <**input**> должен быть использован атрибут **checked.** При передаче данных формы на сервер передаются выражения **name=value** только установленных флажков (**name** и **value** - значения соответствующих атрибутов).

Состояние флажка может быть сохранено следующим выражением:

check_save=document.form1.ch1.checked, где **form1**-значение атрибута **name** формы, а **ch1**-значение атрибута **name** флажка.

Переключатели

Переключатели используются для выбора одного значения из нескольких, составляющих группу. В каждой группе в любой момент времени может быть активизирован только один переключатель. Для выбора другого переключателя достаточно по нему щелкнуть мышью, при этом предыдущий переключатель будет снят.

Для создания переключателей применяется конструкция **<input type="radio">.** В результате создается объект **Radio** со своим набором свойств, методов и событий. Если переключатель установлен, то свойство **checked** объекта **Radio** возвращает значение **true**, в противном случае - **false**. Для установки переключателя по умолчанию в дескрипторе <**input**> должен быть использован атрибут **checked.** Атрибут **name** для группы переключателей имеет одно и то же значение. Группа переключателей представляет собой массив объектов **Radio**, а свойство **length** возвращает значение, равное числу элементов в группе.

Ссылка на отдельный переключатель в группе использует следующий синтаксис:

document.form1.group1[index],

где **form1**-имя формы, **group1**-имя группы переключателей, а **index**-номер переключателя в группе.

Для определения состояния отдельного переключателя нужно воспользоваться выражением:

document.form1.group1[index].checked,

где **form1**-имя формы, **group1**-имя группы переключателей, а **index**-номер переключателя в группе.

Списки

Списки используются для выбора одного значения из нескольких вариантов ответа, если число вариантов достаточно велико. Список представляет собой контейнерный объект **Select**, содержащий элементы **Option.**

Для создания объекта **Select** применяется соответствующий парный дескриптор <**select**>**.** В результате создается объект **Select** со своим набором свойств, методов и событий. Свойство **length** возвращает значение, равное числу элементов списка, а свойство **options** – массив, элементами которого являются элементы списка.

Для создания объекта **Option** служит парный дескриптор <**option**>. Из свойств этого объекта можно выделить свойство **index,** возвращающее индекс элемента массива **options**, **text,** возвращающее строковое значение, заключенное между парой дескрипторов <**option**> и **selected,** имеющее значение **true**, если элемент выбран и **false** в противном случае.

Синтаксис ссылки на объект **Option** с помощью программных средств объекта **Select** следующий:

document.form1.s1.options[index],

где **form1**-имя формы, **s1**-значение атрибута **name** дескриптора **select**, а **index**-номер элемента в массиве **options**.

Для ссылки на выбранный элемент списка (**selectedIndex)** можно использовать следующее выражение:

document.form1.s1.options[document.form1.s1.selectedIndex],

где **form1**-имя формы, **s1**-значение атрибута **name** дескриптора **select**, а **selectedIndex**-номер выбранного элемента в массиве **options**.

Список можно использовать для создания навигационных элементов web-страницы. Каждому объекту **Option** можно присвоить определенный **URL** адрес, а затем воспользоваться объектом **Location** для организации перехода на соответствующую web-страницу. Такой список используется на web -странице, код которой приведен в примере 35

Пример 35

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
function go_to(){
var myList=document.form1.s1;
var myPage=myList.options[myList.selectedIndex].value
```

```
if(myPage!=" " )
{
self.location=myPage;
}
}

</script>
<body>
<form name=form1>
<br>
<select name="s1">
<option value="">Выберите Web-страницу для перехода</option>
<option value="пример5.html">Пример5</option>
<option value="пример6.html">Пример6</option>
<option value="пример7.html">Пример7</option>
<option value="пример8.html">Пример8</option>
<option value="пример9.html">Пример9</option>
</select>
<br><br>

<input type="button" value="Переход" onclick="go_to()">
</form>
</body>
</html>
```

Результат работы кода представлен на рис. 5.9.

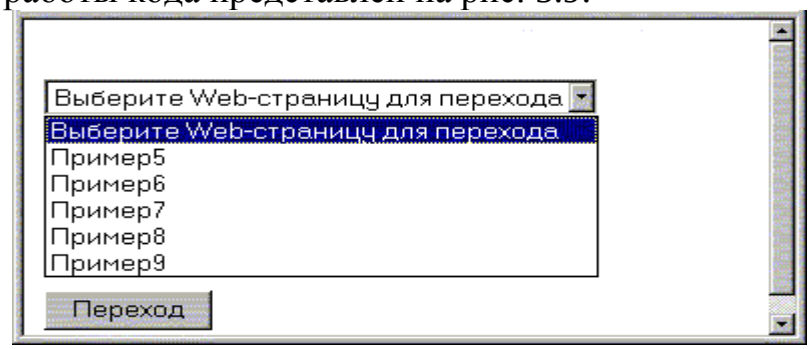


Рис. 5.9

Функцию **go_to()** можно использовать в качестве обработчика события **change** объекта **Select**. В этом случае кнопку "**Переход**" можно исключить из формы. Дескриптор <**select**> в этом случае преобразуется следующим образом:

```
<select name="s1" onChange="go_to()">
```

Для того, чтобы открыть web-страницу не в *текущем* (**self.location=myPage;)** а в *новом* окне браузера, нужно изменить фрагмент сценария:

```
if(myPage!=" " )
{
window.open(myPage);
}
```

Проверка достоверности данных формы

Контроль над значениями полей формы на уровне web-браузера представляет собой одно из самых важных и полезных инструментальных средств JavaScript. Процесс проверки можно представить в виде следующих шагов:

- проверка обязательных для ввода полей;
- проверка типа данных, введенных в каждое поле формы;
- проверка диапазона введенных данных.

Кроме перечисленных шагов можно управлять уровнем проверки данных. Данные могут проверяться на уровне *ввода* (событие **KeyDown**), на уровне *поля* (события **change** или **blur***)* и на уровне *формы* (события **submit** или **click**).

Дополнительные средства, предоставляемые для работы с формой HTML5, позволяют использовать атрибут **required** в обязательных для заполнения полях, устанавливать диапазоны данных в полях, предназначенных для ввода числовых данных и использовать шаблоны для ввода телефонных номеров и другой информации, которая может быть определена шаблоном.

Пример программной проверки заполнения полей перед отправкой, отображения введенных пользователем данных в отдельном диалоговом окне и проверки числовой информации в поле "Ваш возраст" приведен в примере 36, а форма с двумя текстовыми полями и кнопкой – на рис. 5.10.

Проверка на уровне полей осуществляется на наличие в поле возраста числовой информации после потери фокуса данным полем. На уровне формы проверяется заполнение обоих полей данными.

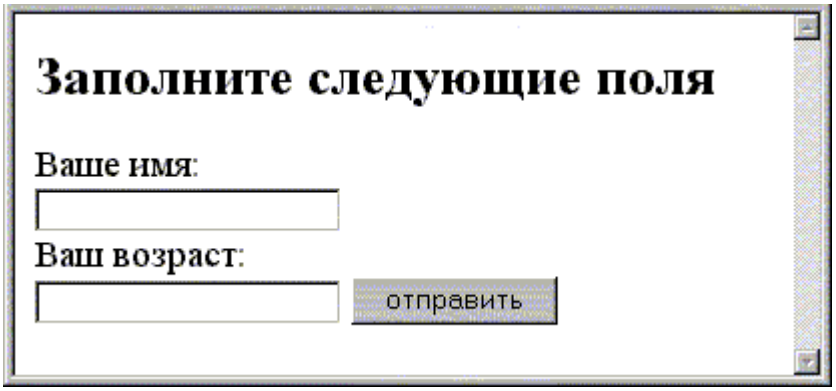


Рис. 5.10

Пример 36

```
<html>
<head>
<script  language="JavaScript" type="text/javascript">
function button_handler(){
var myForm=document.form1;
if(myForm.user_age.value=="" || myForm.user_name.value=="")
{
alert("Пожалуйста, заполните всю форму");
if (myForm.user_name.value=="")
{
myForm.user_name.focus();
}
else
{
myForm.user_age.focus();
}
}
else
{
confirm_message="Вы ввели следующие данные:\n"
confirm_message+="Ваше имя "+myForm.user_name.value+"\n"
confirm_message+="Ваш возраст "+myForm.user_age.value+"\n"
confirm_message+="Хотите отправить данные?"
var submit_ok=confirm(confirm_message)
if (submit_ok){
document.form1.submit()
}
else
{ document.form1.reset()
}}}
function text_handler(){
var age=document.form1.user_age;
if (isNaN(age.value)==true)
{
alert("Введите число");
age.focus();
age.select();
}
}
```

```
</script>
<body>
<h2>Заполните следующие поля </h2>
<form name=form1>
Ваше имя:
<br>
<input type="text" name="user_name" value="Текст по умолчанию">
<br>
Ваш возраст:
<br>
<input type="text" name="user_age" onblur="text_handler()">
<input type="button" value="отправить" onclick="button_handler()">
</form>
</body>
</html>
```

Лекция 11

Глава 6. Библиотека jQuery

Язык JavaScript имеет ряд недостатков, обусловленных тем, что разные браузеры по-разному работают с DOM-моделью, громоздкими вычислительными алгоритмами, в частности, связанными с обработкой массивов, и ограниченными способами обращения к элементам страницы по их идентификаторам (**getElementById**) или по именам (**getElementsByTagName**).

Библиотека jQuery обеспечивает кроссбраузерность, позволяет сделать доступными практически любые элементы страницы, а также их атрибуты, упростить циклические и другие процессы JavaScript и легко реализовать различные визуальные эффекты.

jQuery – это библиотека, основанная на совместном использовании JavaScript, HTML и CSS.

Девиз jQuery – "Write less - Do more!" ("Пиши меньше – делай больше!").

К достоинствам jQuery относится также ее малый объем (50-60 кб), бесплатное использование, большое количество открытого кода, который можно использовать в своих программах.

Для подключения библиотеки можно воспользоваться следующим элементом **<script>**

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js"></script>
```

Схематично процесс создания кода с помощью jQuery можно описать следующим образом:

1. выбор объекта, над которым должны быть произведены действия;
2. определение события, с которым сопряжены действия;
3. описание действий.

6.1 Выбор объекта

После подключения библиотеки программисту доступна функция **Jquery();** или в сокращенном виде **$();**

Анализатор браузера при разборе страницы распознает библиотеку jQuery по этой конструкции.

Синтаксис оператора jQuery можно представить следующим образом:

$('селектор').действие('аргументы'), где селектор – это *объект*, над которым производятся действия, действия – методы, или функции, определяющие, *что именно* будет производиться с объектом, и аргументы – дополнительные параметры действия, если они предусмотрены методом.

Выбор элемента соответствует созданию объекта jQuery, к которому можно будет применять соответствующие методы и использовать соответствующие свойства.

Выбор элементов в jQuery во многом совпадает с выбором селекторов в CSS.

Так, элемент может быть выбран по следующим селекторам:

1. названию элемента **$('p');**
2. классу **$('.main_div');**
3. идентификатору **$('#main_div');**
4. вложенному элементу **$('#main_div tbody tr td');**
5. по перечислению элементов **$('p, div');**
6. следующему элементу **$(' div+ p');**
7. дочернему узлу (прямые наследники) **$('#main_div>img');**
8. по значениям атрибутов
    a. по точному значению **$('img[width=200] ');**
    b. по начальному значению **$('a[href^=http] ');**
    c. по окончанию **$('img[src$=.jpg] ');**
    d. по наличию в любом месте **$('img[src*=foto] ');**
    e. по наличию атрибута **$('div[id]');**
9. с помощью фильтрации
    a. четные **$('#main_table tr:even'); 0 и 2**
    b. нечетные **$('#main_table tr:odd'); 1 и 3**
    c. все, кроме тех, что после not **$('img:not(#div_for_img img) ');**
    d. элементы, содержащие другие элементы **$('div:has(fieldset) ');**
    e. по содержанию текста **$('p:contains(фото) ');**
    f. первый **$(' (#div_for_img:first ');**
    g. последний **$(' (#div_for_img:last ');**
    h. скрытый **$('div:hidden ');**
    i. видимый **$('div:visible');**
10. селекторы форм (выбираются все элементы перечисленных типов)
    a. :input
    b. :text
    c. :password
    d. :radio
    e. :checkbox
    f. :submit
    g. :reset

h. **:button**
i. **:image**
j. **:file**
k. **:checked**
l. **:selected**
m. **:enabled/:disabled**

В один набор можно добавить несколько селекторов через запятую. Например, код применит метод **fadeOut(4000)** к изображениям с шириной **200px** и гиперссылкам, в атрибуте **href** которых есть строка **google**:

```
$('img[width=200], a[href*=google] ').fadeOut(4000);
```

6.2 Действия (методы)

Прежде чем могут быть выполнены какие-либо действия с объектами JavaScript, браузер должен полностью закончить формирование и загрузку DOM-дерева. В jQuery есть механизм, позволяющий автоматически фиксировать полную загрузку DOM-дерева. Он реализован конструкцией **ready(fn),** где **ready** событие, а **fn** – функция обработчика события.

```
$(document).ready(function(){


});// конец ready
```

Или с помощью укороченного варианта:

```
$(function(){


});
```

Многие методы jQuery используются как для задания параметров (в этом случае используется значение аргумента), так и для получения значений (в этом случае аргумент не передается).

Метод **text** позволяет получать и изменять текст элемента.

Методы **hide** и **show** позволяют скрывать и показывать элементы.

Следующий код обеспечивает сохранение в переменной **texth1** текстового значения заголовка **h1** с **id='main_h1',** запись в заголовок измененного текста и вывод результата на экран с помощью метода **alert**:

```
var texth1=$('#main_h1').text();
alert(texth1);
var texth1=$('#main_h1').text('измененный текст');
alert(texth1);
```

Методы **hide** и **show** могут использоваться как без аргументов, так и с аргументом, указывающим время в миллисекундах, которое требуется для сокрытия и показа элементов.

Следующий код обеспечивает сокрытие изображений с атрибутом **src,** содержащим **logo.jpg** и показ их за 3 секунды (3000 мсек).

```
$('img[src*=logo.jpg]').hide(3000);
$('img[src*=logo.jpg]').show(3000);
```

JQuery поддерживает *цепочки* функций, когда к одному объекту можно применить несколько функций, записанных в одной строке кода:

```
var mylogo=$('img[src*=logo.jpg]');
mylogo.hide(3000).show(3000);
```

Методы **fadeOut, fadeIn,** и **fadeTo** обеспечивают плавное исчезновение и появление объекта благодаря изменению параметра прозрачности (**opacity**).

Следующий код обеспечивает исчезновение за 3 секунды **fadeOut(3000),** появление за 3 секунды **fadeIn(3000)** и исчезновение за 3 секунды до заданного уровня прозрачности, равного 0.5 (1 – непрозрачный элемент, 0 – полностью прозрачный) **fadeTo(3000,0.5)** изображений, атрибут **src** которых включает текст **logo:**

```
$('img[src*=logo]').fadeOut(3000).fadeIn(3000) fadeTo(3000,0.5);
```

Методы **slideUp** и **slideDown** обеспечивают исчезновение и появление объекта за счет изменения его высоты (уменьшения – **slideUp** и увеличения – **slideDown**) с параметром, указывающим время в миллисекундах.

Следующий код обеспечивает исчезновение за 3 секунды **slideUp (3000)** и появление за 3 секунды **slideDown (3000)** изображений, атрибут **src** которых включает текст **logo:**

```
$('img[src*=logo]').slideUp(3000).slideDown(3000);
```

Метод **slideToggle(3000)** обеспечивает исчезновение и появление объекта при повторном обращении к нему.

6.2.1 Методы для работы с CSS

Метод **css** дает возможность работы с CSS-стилями.

Добавить стиль объекту можно, используя следующий код:

```
$('#div_table').css('color', '#3f4078');
```

Если нужно добавить несколько правил CSS, то можно использовать цепочку функций:

```
$('# div_table').css('color', '#3f4078').css('font-size','19px');
```

или использовать следующую конструкцию:

```
$('# div_table ').css({
'color': '#3f4078',
'font-size':'19px'
)};
```

Добавить класс можно с помощью метода **addClass('class')** и удалить класс с помощью метода **removeClass('class').**

Методы **width() и height()** позволяют считывать и задавать ширину и высоту объекта.

Следующий код считывает в переменные **divW** и **divH** ширину и высоту элемента **div** и присваивает ему новые значения:

```
var divW=$('div').width();
var divH=$('div').height();
var divW=$('div').width(500).height(100);
```

6.2.2 Методы для работы с HTML

Метод **html** позволяет считывать HTML-код, например, элемента с идентификатором **#div_head':**

```
alert($('#div_head').html());
```

или изменить существующий код:

```
$('#div_head').html('<h1>Новый заголовок</h1>');
```

Методы **append(), prepend(), after(), before()** позволяют добавить html-код в элемент *после* существующего контента - **append(),** *до* контента внутрь элемента - **prepend(),** *после* элемента - **after()**, *до* элемента - **before().**

Методы **clone() и remove()** позволяют клонировать и удалять элементы:

```
// клонируем объект с идентификатором div_for_img и сохраняем его в переменной mm
var mm=$('#div_img ').clone();
// после объекта с идентификатором myform поместим этот клонированный объект
$('#myform').after(mm);
// удалим объект с идентификатором myform и сохраним его в переменной nn
var nn=$('#myform').remove();
//перед объектом с идентификатором mtable вставим этот объект
$('#mtable').before(nn);
```

Методы работы с атрибутами **attr() и removeAttr()** позволяют считывать, изменять и добавлять значения атрибутов, а также удалять их:

```
//считываем значение атрибута height элемента img
var mh=$('img').attr('height');
//присваиваем значения атрибутам во всех элементах img
$('img').attr({src:'images/pict.gif',alt:'рисунок'});
//присваиваем атрибуту height значение 200px
$('img[height]').attr('200px');
// присваиваем атрибуту size элемента с идентификатором m_sel значение 4
$('#m_sel ').attr('size',4);
```

//удалим атрибут **title** элемента с идентификатором **pict** и присвоим ему новое значение

```
$('#pict').removeAttr('title').attr('title','новое значение');
```

6.2.3 Методы для создания анимации

Метод **animate()** позволяет изменить свойства, которые имеют числовые значения, за какой-то промежуток времени, что создает эффект анимации:

//изменение свойства **paddingTop** до значения 450px за 4000мсек

```
$('#main_h1').animate({'paddingTop' : '450px'},4000);
```

//изменение свойства **width** до значения 400px за 4000мсек, затем работает функция с выводом сообщения

```
$('#div_for_img
').animate({'width':'400px'},4000,function(){alert('сообщение')});
```

Метод **each** и **$(this)**

Для обращения ко всем элементам выборки используется метод **each**, а для обращения к каждому элементу – функция **$(this).**

Код позволяет применить к элементам **img** метод **fadeOut(4000),** если их высота меньше 100px:

```
$(‘ img’).each(function(){
If($(this).height()<100){
$(this).fadeOut(4000);
}
});
```

6.3 Автоматические циклы

JQuery позволяет заменить циклические процессы, которые используются в JavaScript, простой строкой кода. Следующий код скрывает все изображения, являющиеся дочерними, элемента с идентификатором **div_for_img:**

```
$('#div_for_img  img').hide(3000)
```

6.4 События и их программирование

Браузер фиксирует все действия пользователя и выполняет код, который является обработчиком события. В jQuery при наступления события должен выполниться код:

объект.событие(функция обработчика события);

Такая функция может не иметь имени и называется анонимной функцией.

Список событий приведен далее:

mouseover – наведение мыши,

mouseout – сведение мыши,
click – щелчок мыши,
dblclick – двойной щелчок мыши,
mousemove – перемещение мыши,
mousedown – нажатие левой кнопки мыши без отпускания,
mouseup – отпускание кнопки мыши,
submit – отправка формы,
focus – фокус на элементе,
blur – потеря фокуса,
change – изменение состояния элемента,
reset – сброс значений элементов управления формы,
keypress – нажатие клавиши (состоит из двух следующих):
keydown,
keyup,
load –загрузка браузером всех объектов страницы,
resize – изменение размера окна,
scroll – прокрутка страницы,
unload – закрытие страницы или переход на другую.

Следующий код позволит вывести окно предупреждения с помощью метода **alert** при щелчке на дочернем изображении элемента с идентификатором **div_for_img:**

```
$('#div_for_img  img').click(function(){
alert('сообщение');
});
```

При прокрутке страницы выполняются действия, прописанные в теле функции:

```
$(window).scroll(function(){
Тело функции
});
```

Следующий код позволит скопировать изображения из одной области страницы (с идентификатором **div_for_img**)и добавить ее в другую (с идентификатором **div_for_result**):

```
$('#div_for_im   img').click(function(){
var mm=$(this).clone();
$('#div_for_result').append(mm);
});
```

Следующий код обеспечит подсветку строки таблицы, на которую наведена мышь и снимет ее при сведении мыши с этой строки. Стилевые свойства **(background-color и color)** записаны в класс с именем **class1**:

```
var tabstr=$('#div_table   tr')
tabstr.mouseover(function(){
```

```
$(this).addClass('class1');

});
tabstr.mouseout(function(){
$(this).removeClass('class1');
});
```

В jQuery используются события **hover** и **toggle**. Событие **hover** заменяет события **mouseover** и **mouseout**, а событие **toggle** заменяет последовательные события **click**.

Для предыдущего примера используем событие **hover**. Код приведен далее:

```
$('# div_table   tr').hover(function(){
$(this).addClass('class1');
},function()
{
$(this).removeClass('class1');

});
```

Следующий код позволит свернуть форму при щелчке по ее заголовку и в нем записать текст 'развернуть форму', а при повторном щелчке - развернуть форму и в заголовке записать текст 'свернуть форму':

```
$('#form_h').toggle(function(){
$('#my_form').fadeOut(1000);
$(this).text('развернуть форму');
},function(){
$('#my_form').fadeIn(1000);
$(this).text('свернуть форму');

});
```

6.4.1 Объект события

Информация о событии сохраняется в объекте события. Для получения доступа к объекту события в анонимную функцию передается параметр с произвольным именем. При наступлении события в эту переменную занесется вся информация (свойства) об объекте события.

В частности, будут доступны свойства объекта **screenX, screenY** - расстояния по горизонтали и вертикали от угла монитора, **pageX, pageY** - расстояния по горизонтали и вертикали от угла браузера, **altKey, ctrlKey, shiftKey** – были ли нажаты соответствующие клавиши в момент наступления события (возвращается значение **true** или **false**), **target** – на каком объекте произошло событие.

Следующий код использует параметр **eventObj** для доступа к объекту события и выводит на экран координаты щелчка мыши на объекте по горизонтали и вертикали, в переменной **my** сохраняется объект и выводится на экран его идентификатор.

```
$('#div_for_img').click(function(eventObj){
alert( eventObj.screenX);
alert( eventObj.screenY);
var my= eventObj.target;
alert(идентификатор объекта события' + my.id);});
```

6.4.2 События форм

Отправка на сервер осуществляется при наступлении события **submit**. Следующий код осуществляет проверку заполнения текстового поля с идентификатором **t1_mail** перед отправкой данных формы с идентификатором **my_form**:

```
$('#my_form').submit(function(eventObj){
If  $('#t1_email').val()==""{
eventObj.preventDefault();
alert('поле не заполнено');
}
});
```

6.4.3 Активное и запрещенное состояние полей форм

Атрибут **disabled** блокирует поле. Следующий код блокирует элемент управления с идентификатором **b1**, если из списка с идентификатором **my_select** выбран элемент со значением, равным 3:

```
$('#my_select').change(function(){
Var my=$('# my_select :selected').val();
If (my==3){
$('#b1').attr('disabled', 'disabled');
}
Else {
$('#b1').attr('disabled', false);
}
});
```

6.5 Отмена стандартного действия

Для отмены стандартного действия, например, перехода по ссылке, нужно использовать конструкцию **preventDefault.**

Следующий код отменит действия, происходящие при щелчке по гиперссылке с идентификатором **suplink:**

```
$('#superlink').click(function(eventObj){
eventObj. preventDefault();});
```
Вместо **eventObj. preventDefault()** можно использовать **return false.**

Лекции 12, 13

Глава 7. Создание электронного представительства (сайта) предприятия

Название "сайт" (англ. **site** – площадка, место для размещения) утвердилось взамен терминов "узел" или "сервер". Сайт представляет собой *информационную единицу* в отличие от сервера (*программного обеспечения*) или узла Интернет - (*компьютера*).

Сайт – это набор web-страниц, связанных единой темой, общим оформлением, взаимными гипертекстовыми ссылками и, обычно, компактным размещением.

С помощью ссылки можно попасть из сети на любую страницу сайта, поэтому каждая страница должна представлять собой цельную композицию, в особенности первая, на которой знакомство с сайтом может быть закончено.

Еще одна отличительная особенность сайта – нелинейность. В отличие от книги, чтение которой происходит от страницы к странице, сайт может быть построен так, что позволяет перейти от одного предложения или абзаца к другому, удаленному от первого на несколько страниц или, напротив, находящегося вблизи него. Это значит, что отдельная страница не имеет фиксированного положения.

В то же время между страницами существуют связи, которые определяют структуру сайта и отражают единство содержимого. Можно выделить два основных типа таких структур – древовидные (иерархические) и линейные (последовательные). Древовидная структура более всего подходит для разнородного и сложно устроенного материала – каталогов, сборников статей и т.д. Если же материал выстраивается в последовательную цепочку, естественно воспользоваться линейной структурой. Такая структура используется для глав книги или шагов оформления заказа в электронном магазине. Большинство сайтов используют и древовидную, и линейную структуры.

Возможность генерировать страницы в ответ на запросы пользователя размывают стройную картину структуры сайта, так что на многих современных сайтах невозможно сказать, из скольких страниц они состоят и сколько связей содержат.

Однако при создании собственного сайта, нужно четко представлять и его структуру, и совокупность ссылок, при этом особенно важно выделить внешние ссылки, связывающие сайт с другими.

При планировании структуры сайта можно пользоваться различными способами - от раскладывания карточек, символизирующих страницы сайта на столе и использования эскизов до организационных диаграмм.

Web-редакторы, такие, как Macromedia Dreamweaver, имеют в своем составе средства, позволяющие создавать карту сайта. Карта сайта позволяет моделировать структуру сайта и связи между его страницами.

7.1. Разновидности сайтов

Самая массовая категория web-сайтов–это личные страницы. Web дает огромные возможности для самовыражения и популяризации личности и является уникальным местом для реализации этих возможностей. Успех личного сайта зависит от многих причин, но на первое место следует поставить осмысленность цели создания и ценность его содержания.

Еще одна категория – это сайты некоммерческие, принадлежащие всевозможным добровольным объединениям, международным или благотворительным организациям. К этой категории можно отнести и многочисленные сайты учебных заведений, университетов и научных центров. Такие сайты характеризуются обычно логичным и последовательным оформлением.

Самой заметной разновидностью сайтов являются контент-сайты, основные поставщики пользующейся спросом информации. К этой категории относятся поисковые системы, сайты новостей, развлекательные и образовательные ресурсы. У сайтов этого типа содержание превосходит по важности оформление, поэтому уровень дизайна таких сайтов ниже, чем у сайтов коммерческих.

Достаточно распространенной категорией сайтов являются представительские сайты предприятий.

7.2 Проектирование представительского сайта предприятия

Представительские сайты являются одновременно и информационными, и рекламными. Для них характерен продуманный, часто профессионально выполненный дизайн, соблюдающий принципы единства оформления сайта и отсутствия на нем лишних элементов. Однако следует помнить, что дизайн любого сайта, а в особенности представителдьского – это, прежде всего "информационный" дизайн, который должен отдавать предпочтение содержанию над оформлением.

Посетитель такого сайта должен получить представление как о самом предприятии (фирме или магазине), так и о продукции, выпускаемой фирмой. Сведения о товаре и о фирме составляют информационную часть сайта.

В результате посещения такого сайта у посетителя должен создаться образ фирмы – марка и логотип служат именно этой цели. Марка продукции должна связывать изображение или текст с теми свойствами, которые характерны только для данной продукции, например, качеством, упаковкой, сопутствующими услугами, гарантией и т.д. Узнаваемый и

оригинальный логотип позволяет лучше запомнить фирму, ее продукцию или услуги.

Можно выделить следующие основные задачи, встающие перед проектировщиками представительских сайтов:

1. Определение долгосрочных и краткосрочных целей создания сайта:

-распространение информации о товарах, услугах и фирме;

-формирование имиджа фирмы и марки продукции;

-обеспечение непрерывного предоставления услуг клиентам и партнерам;

-установление новых связей и поддержка старых;

-обеспечение информационной и сервисной поддержки клиентов;

-продвижение продаж.

2. Анализ рынка с целью определения целевой аудитории и конкурентов:

-определение потенциальных клиентов, их запросов, наличия телекоммуникационного оборудования;

-определение способов привлечения новых клиентов и удержания старых;

-анализ существующих сайтов по данной тематике, их достоинств и недостатков.

3. Экономическое обоснование:

-какие средства требуются для создания сайта (разработку, подготовку материалов, размещение и поддержку);

-каков положительный результат создания сайта (увеличение прибыли, улучшение имиджа);

-каков круг лиц, отвечающих за создание и поддержку сайта.

4. Проектирование информационно-сервисной архитектуры сайта:

-каталоги товаров или услуг;

-прайс-листы;

-навигационная система;

-информация о фирме;

-анонсы (скидки, акции, информация о новых товарах или услугах);

-анкеты, формы, форумы и т.д. для обратной связи с клиентами;

-поиск по сайту;

-регистрация для дальнейшей персонификации пользователя;

-формирование FAQ;

-возможность использование CRM-системы;

-проектирование логической структуры сайта (определение разделов и их связи);

-проектирование физической структуры сайта (распределение материалов по файлам и папкам);

-макетирование:

-создание эскизов страниц;

-проработка дизайна страниц;

-выбор средств создания страниц (CSS, JavaScript, табличный дизайн);

-выбор программной платформы и возможности использования CMS.

5. Создание контента (разработка и заимствование) и структурирование текстовых, графических и других материалов, составляющих информационную, сервисную и оформительскую (графические файлы, файлы мультимедиа и т.д.) части сайта.

6. Тестирование:

-оценка результативности (получение ожидаемого результата);

-оценка правильности функционирования (навигационной системы, времени загрузки);

-оценка usability (удобства использования, практичности);

-оценка привлекательности дизайна.

7. Размещение:

-на собственной телекоммуникационной платформе;

-с привлечением сторонних средств.

8. Продвижение и поддержка:

-регулярное обновление информационной и сервисной частей сайта;

-регистрация в поисковых системах и каталогах;

-размещение направленной рекламы и участие в партнерских программах;

-использование SEO (Search Engine Optimized, оптимизация под поисковые машины);

-использование web-аналитики.

7.3 Принципы создания пользовательского интерфейса

Дизайн web-страниц подчиняется правилам создания пользовательского интерфейса, который основан на следующих принципах.

Использование метафоры. При этом элементы сайта имеют символическое представление. Обычно они представлены знакомыми объектами и служат "визуальной помощью" для ориентации по сайту или выбору ссылок. Пример использования визуальных метафор можно увидеть на сайтах Yahoo!, где для переходов предлагаются пиктограммы с легко узнаваемыми рисунками (рис. 7.1). Хорошо известным примером использования метафоры является "Рабочий стол" или "Папка" в операционной системе Windows, которые заменяют реальные физические и логические объекты.

Рис.7.1

Ясность компонентов страниц сайта. Недопустимо использование абстракции в элементах навигации или в информационных частях

страницы, хотя абстракция как художественный прием уместна, например, при создании логотипа.

Последовательность в оформлении отдельных страниц сайта, или единство оформления – один из важнейших принципов дизайна. Страницы могут быть связаны единым фоном, цветовой гаммой, шрифтами, навигационными панелями, повторением логотипа и элементов управления, с помощью которых происходят переходы внутри сайта на каждой странице, а также использованием подобно оформленных блоков текста.

Поддержка ориентации и навигации. Ориентация означает, что пользователь должен в любой момент знать, в каком именно месте сайта он находится. Ориентация достигается, например, созданием заголовков страниц или выделением навигационного элемента, соответствующего текущей странице. Кроме того, посетитель сайта должен иметь возможность вернуться на первую и соседние страницы. Если страница слишком длинная (рекомендуется ограничить ее длину двумя – тремя экранами), то нужно предусмотреть возможность возврата в ее начало.

Навигация по сайту должна быть удобной и простой. Желательно ограничить число переходов для попадания в любую часть сайта.

Список навигации с описанием ссылок или графическое представление файлов сайта и их взаимных связей используется как карта сайта. Карта помогает видеть сайт в целом и может использоваться для тестирования сайта.

7.3.1 Макетирование страниц

Обычно на web-сайте различаются первая страница, информационные страницы и страницы обратной связи. Кроме этого могут понадобиться файлы мультимедиа и другие специальные средства.

HTML-текст каждой страницы не должен превышать 20-30 Кб.

Первая страница имеет особое значение, поэтому ее дизайн заслуживает пристального внимания. Первая страница должна быть одновременно привлекательной и информационной. Эта страница должна быть оформлена в сочетании с остальными страницами и в то же время выделяться более остальных. Представительские сайты обычно используют для первой страницы яркую, тщательно подобранную фотографию или рисованную графику, иллюстрирующую деятельность фирмы, ее главный продукт или важнейшую из последних новостей. Это изображение – визуал становится центром композиции.

Информационные страницы должны не только удовлетворить спрос посетителя по существу вопроса, но и заставить обратить внимание на интересное оформление и сохранить интерес к каждой новой странице.

Формы обратной связи нужно рассматривать как часть проекта. При работе с формой важно сохранять цветовое оформление, шрифты и другие особенности композиции.

Первым шагом макетирования страницы служит создание эскизов, которые можно выполнить либо вручную, либо с помощью графической программы, например, слоев PhotoShop.

После создания эскиза нужно переходить к средствам макетирования, реализуемым с помощью HTML и CSS. Основная задача при макетировании - позиционирование (размещение) элементов страницы. Макетирование страниц можно выполнять разными способами.

Распространенным способом является использование таблиц, однако, более предпочтительным является использование позиционирования с помощью CSS. Применение таблиц позволяет осуществить динамическое изменение размеров таблицы, которые изменяются в зависимости от размера экрана или окна браузера.

Динамические таблицы не ограничены рамками фиксированных размеров. Для описания такой таблицы ее размеры указываются в процентах по отношению к окну браузера. При изменении окна таблица изменит свои размеры. Недостатком такой таблицы является невозможность обеспечить точные размеры ее составляющих.

Кроме таблиц для макетирования применяются фреймы, хотя их применение с появлением JavaScript и CSS значительно сократилось.

Кроме фреймов существуют еще способы вывода на экран компьютера нескольких файлов одновременно – это создание новых окон браузера.

Новые окна могут быть созданы с помощью атрибута **target** дескриптора **<a>** для документа, к которому ведет эта ссылка. Документ может быть загружен либо в новое окно, либо в одно из уже открытых окон.

Наконец, JavaScript позволяет открывать новые окна с указанием их размеров и режимов, запрещающих их изменение.

7.3.2 Основы и принципы дизайна [4]

7.3.2.1 Пространственные отношения

Любая web-композиция строится на основе пространственных отношений, которые определяются размерами составляющих ее элементов, их взаимным расположением и пропорциями (соотношением размеров элементов).

При выборе размера элемента нужно учитывать различные факторы, влияющие на восприятие размера, такие как вклад данного элемента в информационное содержание страницы, положение в композиции, а также цвет и форму.

Так, чем контрастнее элемент на фоне его окружения, тем он кажется бОльшим. Теплые тона привлекают внимание к элементу, делают его более близким, активным и по субъективному восприятию бОльшим, холодные тона удаляют и уменьшают объект.

Наконец, при выборе размеров объектов страницы нужно помнить, что слишком большие их размеры могут привести к появлению горизонтальной полосы прокрутки, что нарушит композицию страницы. При размере экрана 640×480 пикселов на экране можно занять только 585×295 пикселов. При других размерах экрана (800×600, 1024×768, 1280×1024, 1600×1200) полезная площадь также уменьшена (745×415, 969×583, 1225×839). Если композиция создавалась для экрана с меньшим размером, чем экран посетителя, то объекты будут выглядеть мельче и появится свободное пространство, в противном случае неизбежно появление нежелательной горизонтальной полосы прокрутки.

При размещении элементов на странице большое значение имеет выравнивание. Для текста это применение различных видов выравнивания (по центру, левостороннего, правостороннего или по ширине), а также межстрочных расстояний и расположения первой строки абзаца. Блоки текста могут быть выровнены с помощью предваряющих их маркеров, выполняющих роль точек привязки.

Для разноуровневых текстовых элементов (заголовка и блоков текста) часто применяют выдвинутые влево относительно текста заголовки, а также заголовки, традиционно расположенные по центру.

Нетекстовые элементы должны иметь явную (прорисованную) или неявную линию выравнивания. Линия выравнивания становиться заметной, если расстояния между элементами не слишком малы и не слишком велики.

Соотношение размеров элементов (пропорции) чаще всего имеют значения 1:1, 1:2 или 1:3. Соотношение между элементами. очень важно для создания визуальной гармонии.

Важным является наличие свободного пространства, которое является элементом дизайна и дает возможность сформировать определенную структуру страницы. Ключевые элементы страницы должны быть окружены достаточным свободным пространством. Применение полей при работе с большими блоками текста позволяет зрительно уменьшить ограниченность компьютерного экрана. Простым способом установки полей является использование соответствующих свойств CSS.

7.3.2.2 Форма

Форма - отличительное свойство элемента страницы. Объекты страницы в зависимости от формы можно разделить на три группы:

-построенные из прямых линий и углов (горизонтальные, вертикальные прямые и прямоугольники),

-содержащие криволинейные формы (дуги, окружности),

-аморфные (бесформенные) объекты.

Прямые линии выполняют функции разделителей и соединителей. Разделительные линии применяются достаточно часто, внешний вид их изменяется от стандартных линий, создаваемых с помощью дескриптора **<hr>** до разнообразных графически созданных изображений.

Соединительные связующие линии используются для придания композиции целостности и могут выполнять функции линий выравнивания. Разновидностью соединительных линий являются прорисованные базовые линии текста, особенно уместного в заголовках. Соединительные линии могут использоваться для создания связи подписи с иллюстрацией, кнопки навигации с соответствующим изображением, заголовка с текстом и т.д.

Самой частой формой, используемой в web-страницах, является прямоугольник. Прямоугольник обычно используется с внесенным в него текстом. При этом текст не должен плотно прилегать к контуру (рис. 7.2а), между текстом и контуром нужно оставлять достаточно места, чтобы текст не выглядел зажатым в границы прямоугольника. Если используется заливка прямоугольника фоновым цветом, то контур можно не прорисовывать (рис. 7.2б), в крайнем случае сделать его утолщенным (рис. 7.2в,г).

(а) (б) (в) (г)

Рис. 7.2

Закругления довольно часто используются на web-страницах, хотя окружность слишком противоречит прямоугольности компьютерного экрана и листа бумаги и редко является основой композиции. Использование дуги, закругляющей острые углы прямоугольников, является распространенным приемом. Однако, этот прием требует закругления и других углов элементов, участвующих в композиции.

Аморфные элементы служат для контраста геометричности композиции. Аморфные элементы могут быть созданы в графических редакторах с помощью специальных фильтров. Одним из видов аморфных элементов являются фракталы - рекурсивные формы, части которых повторяют сами себя.

7.3.2.3 Цвет

Цвет принадлежит к базовым строительным материалам в web-дизайне и нередко становится основой композиции. Экран компьютера, излучающий свет, позволяет охватить гораздо более широкий цветовой спектр, чем тот, который можно воспроизвести на бумаге.

Цвета в окружающем мире создаются природными красителями– веществами, реагирующими на свет. При попадании на них света, эти

вещества поглощают и отражают от себя конечный результат, который и воспринимается как цвет. Цвет в природе – это субтрактивный (вычитающий) цвет.

Компьютер отображает цвета иначе. Компьютерное отображение цвета происходит на основе аддитивного синтеза – сложения трех основных цветов: красного, зеленого и синего. При варьировании этих компонент и излучаемого света можно получить множество цветов, применяемых в web-страницах.

Цветовые возможности компьютера зависят от его видеопамяти, в которой храниться экранное изображение. В зависимости от числа бит, приходящихся на пиксел, изменяется такая характеристика, как глубина цвета. Если эта характеристика находится в интервале от 1 до 8 бит, то компьютер может отобразить от 2 до 256 (2^8) цветов. Уровень качества, при котором глаз не способен отличить компьютерную фотографию от настоящей достигается при не менее чем 24 битах на пиксел. При этом воспроизводится около 16 млн. цветов. Такой режим цветопередачи называется true color. Однако существуют мониторы, которые могут работать в режиме high color, обеспечивающем 15 бит на пиксел, т.е. около 30 тыс. цветов. Этот режим для большинства практических нужд не уступает true color.

7.3.2.4 Системы представления цвета (рис. 7.3)

Самая распространенная система представления цвета – это RGB (Red, Green, Blue). В HTML числовые значения RGB представлены в шестнадцатеричной системе счисления (0-9,A,B,C,D,E,F). Каждая составляющая содержит пару шестнадцатеричных цифр, таким образом, каждый цвет может быть представлен 8 битами, а все значение - 24 битами, что соответствует максимальной глубине цвета, например, 33FFCC.

Цвет на компьютерном дисплее изменяется от черного (отсутствие цвета) до белого (максимальная яркость всех трех составляющих), а на бумаге, наоборот, отсутствию цвета соответствует белый, а смешению всех красок - черный. Поэтому вместо аддитивной системы RGB при подготовке к печати изображение должно быть переведено в субтрактивную (вычитающую) систему, использующую противоположные исходным цвета – циан (голубой), пурпурный и желтый. Для расширения диапазона цветовоспроизведения к этим компонентам добавляется черный, а вся система получила название CMYK (Cyan, Magenta, Yellow, BlacK).

В компьютерных графических программах применяется система представления цвета HSV (Hue-тон, Saturation-насыщенность, Value-яркость). Эта система моделирует восприятие цвета человеком, а не его физические свойства.

Тон – это один из цветов цветового круга - совокупности цветов, воспроизводимых компьютером.

Насыщенность определяется присутствием в основном тоне доли серого цвета, который при максимальной насыщенности отсутствует вообще, а при минимальной насыщенности заменяет собой основной тон.

Яркость при своем максимальном значении превращает любой цвет в белый, а при минимальном – в черный.

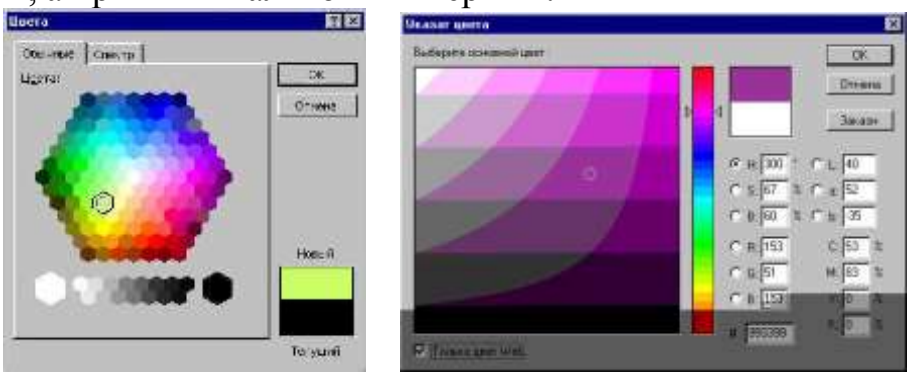


Рис .7.3

В цветах на цветовом круге можно выделить самостоятельные и переходные цвета. Так, красный, синий и зеленый являются самостоятельными и контрастными цветами. Рядом с самостоятельными цветами находятся похожие цвета, которые можно использовать для создания цветовой гаммы. Наконец, цвета нижней половины круга (если синий цвет расположить в его нижней точке) считаются холодными цветами, а цвета верхней половины круга считаются теплыми. Принято считать также, что теплая окраска приближает объект, делает его визуально больше и активнее, а холодный цвет отдаляет, успокаивает, переводит объект на задний план композиции.

7.3.2.5 Безопасная палитра

Несмотря на привлекательные возможности режимов true color и high color, большой процент пользователей сети применяют палитру, состоящую из 256 цветов. Однако для корректно отображаемой палитры этих цветов еще меньше - 216. Палитра из 216 цветов называется безопасной палитрой и отображается без редуцирования (замены отсутствующих цветов на наиболее подходящие) всеми компьютерами на всех платформах. При работе с корректно отображаемыми цветами нужно помнить, что их значения в системе RGB должны быть составлены только из комбинаций 00, 33, 66, 99, CC, FF.

7.3.2.6 Выбор цвета для элементов html-документа

Текст, фон и гиперссылки – это элементы web-страницы, для которых необходимо выбрать подходящие цвета. Рекомендуется выбирать цвета из безопасной палитры и таким образом, чтобы спектр отдельной страницы не превышал трех - четырех цветов.

Подбор цветов должен удовлетворять принципам единства и контраста. Принцип единства требует, чтобы цвета были как можно ближе

друг к другу или являлись бы оттенками основного тона. Контраст же подразумевает различие между цветами. Добиться контраста можно, изменяя одну из составляющих контрастирующих цветов, лучше, яркость, оставляя при этом остальные составляющие (тон и насыщенность) одинаковыми.

Простое решение может быть достигнуто выбором минимального количества контрастных цветов.

Контрастные цвета обычно разделены четвертью цветового круга. Можно использовать контраст теплых и холодных тонов. Основное требование к паре цветов "текст-фон" – достаточный контраст между ними. Используют страницы с темным текстом на светлом фоне и со светлым текстом на темном фоне.

Хорошо смотрятся в качестве фона насыщенные холодные цвета, такие, например, как глубокий синий. Для текста популярны оттенки теплых цветов, в особенности желтого.

При использовании заголовков разных уровней можно подчеркнуть вложенность заголовка применением более светлого оттенка тона основного заголовка.

В качестве фона иногда применяются текстуры (рис. 7.4), которые имитируют реальные поверхности – деревянные, мраморные, кирпичные. Текстура требует тщательной проработки композиции и координации с другими элементами, при этом недопустимо использовать несколько текстур на странице, да и остальная графика должна быть сведена к минимуму. Кроме "материальных текстур" используют "геометрические текстуры", которые состоят из геометрических узоров – полос, ромбов, клеток, а также "фотографические" текстуры, в качестве которых используются фотографии в сочетании с "размытостью". Такие фотографические текстуры обычно используют вместе с плоским (однородным) фоновым цветом.

Рис. 7.4

Ссылки должны быть более заметны, чем текст, и этого можно добиться контрастом тона, увеличением насыщенности или понижением яркости. Посещенные ссылки должны иметь цвет менее насыщенный, чем непосещенные или быть по тону более близкими к основному тексту.

7.3.2.7 Шрифты

Использование шрифтов является важным средством визуального проектирования. Шрифты для Web могут быть традиционными, т.е. встроенными в операционную систему компьютера, могут быть созданы в графических программах или быть внедренными, т.е. передаваться вместе с html-страницей и автоматически загружаться на компьютере пользователя, подобно графическим объектам.

Все существующие шрифты можно разделить на несколько категорий (рис. 7.5):

Serif – шрифты с засечками, или серифами (Times, Garamond, Georgia, Century Schoolbook).

Sans-serif – шрифты без засечек, или рубленые (Helvetica, Arial, Verdana).

Monospaced – моноширинные, каждая буква в таком шрифте имеет одинаковую ширину (Courier, Courier New).

Decorative – декоративные шрифты, к ним относятся различные стилизации (готический, древнерусский, рукописные (каллиграфические или нарочито небрежные) (Whimsy, Nuptial Script).

В пределах одной категории нужно различать гарнитуру шрифта – набор начертаний одного шрифта.

Из начертаний внутри одной гарнитуры чаще всего используют прямое (regular) и курсивное (наклонное) (italic) начертание, а для некоторых рубленых шрифтов слегка перекошенное вправо (oblique).

Кроме наклона начертание характеризуется еще и насыщенностью, которая характеризует толщину штрихов в рисунке букв. Эта характеристика может принимать значения "жирный шрифт" (bold), "полужирный" (demi-bold), а также светлый (light), и сверхжирный (extra-bold). Эти характеристики, в отличие от курсива, удобнее менять у рубленых шрифтов.

Рис. 7.5

Для некоторых шрифтов существуют особые варианты представления – сжатые (condenced) или растянутые (expended). Именно такими вариантами шрифтов нужно пользоваться при желании изменить площадь, занимаемую надписью, в противовес изменению шрифта в графической

программе, которое приведет к искажению исходного рисунка букв и нарушению пропорций.

Параметр, характеризующий размер шрифта, называется кегль (font-size). Обычно, размер шрифта измеряется в пунктах (0,35мм).

Кроме основных характеристик можно назвать также интерлиньяж (line height) – расстояние между строками текста и кернинг – регулировку расстояния между буквами в слове.

HTML предоставляет ограниченные средства управления шрифтами, которые сосредоточены, в основном, в дескрипторах создания заголовков <h1>…<h7>.

В соответствии с рекомендациями консорциума W3C характеристики шрифтов должны быть заданы с помощью шрифтовых свойств CSS.

Подбор шрифтов на web-страницах должен удовлетворять тем же принципам единства и контраста, что и подбор цветов. Композиция должна содержать минимальное количество резко различных, контрастирующих шрифтов. Лучшей парой шрифтов является шрифт с засечками для основного текста и рубленый шрифт для заголовков. Хорошо смотрится также вариация этих шрифтов – курсив у шрифтов с засечками и жирное или сверхжирное начертание у рубленого шрифта.

Использование декоративного шрифта возможно только при единственности этого шрифта в композиции, т.к. подобрать сочетание к такому шрифту достаточно сложно и, возможно, парой к нему может быть только простой рубленый шрифт типа Helvetica.

Моноширинные шрифты, такие как Courier хорошо контрастируют с фотографическими текстурами и размытыми графическими изображениями.

Размер шрифта имеет для страницы большое значение. Обычно, заголовки верхнего уровня используют шрифт большего размера, чем основной текст и вложенные заголовки.

7.3.2.8 Основные принципы дизайна

Принципы, которые лежат в основе дизайна - это единство (целостность), баланс (уравновешенность), контраст и динамика.

Первый и основной принцип дизайна – принцип единства. Этот принцип требует экономии средств, необходимости обходиться тем, что уже введено в композицию страницы. Он предписывает не вводить новые элементы, служащие лишь "украшением" страницы и не несущие достаточной смысловой нагрузки.

Объекты, выполняющие одни и те же функции, должны быть оформлены одинаково.

В размещении принцип единства сводится преимущественно к выравниванию элементов, т.е. к уменьшению числа вертикалей и горизонталей на странице.

Использование форм в соответствии с принципом единства требует отсутствия их многообразия и, соответственно, наличия форм какого-нибудь одного типа (скругленных, прямолинейных или аморфных).

Цвета на странице должны быть подобраны так, чтобы ограничиться парой контрастных цветов, не считая черного и белого.

В соответствии с принципом единства, лучше ограничиться двумя шрифтами – с засечками для текста и рубленым для заголовков. В случае же использования декоративного шрифта, его желательно оставить на странице в одиночестве.

Баланс - это соразмерность и уравновешенность, в частном случае - симметрия композиции. Понятие баланса тесно связано с плотностью размещения материала. В частности, композиции могут различаться по информационной плотности, плотности цветовой или шрифтовой. Можно предположить, что гармоничными окажутся те композиции, в которых распределение плотности в разных аспектах не совпадает, но скоординированы друг с другом.

Контраст придает композиции живость и лежит в основе производимого впечатления web-страницы.

Проявление контраста в размере лучше всего воспринимается, если разница размеров не слишком велика и не слишком мала.

Контраст в размещении можно получить, например, установив иерархические отношения вложенности элементов.

Контраст цветов достигается соответствующим подбором яркости, насыщенности или тона контрастирующей пары.

Форма является средством для достижения единства, а не контраста.

Динамика служит для придания композиции движения (движения восприятия).

Неявная динамика присутствует при чтении текста, перемещении взгляда по ссылкам, по цветовым пятнам, по навигационным панелям. Элементы, которым присуща неявная динамика, могут использовать стрелки, линии с одним "глухим" концом, на котором может находиться надпись или другой статический объект.

Явная динамика достигается помещением на страницу какого-либо движущегося объекта, например, автомобиля или бегущего человека. Это может быть не только фотография, но и изображение с видимостью движения, созданное в графических редакторах с помощью специальных фильтров, например, Motion Blur в Photoshop.

И, наконец, можно поместить на странице анимированное изображение, что является, однако, скорее принадлежностью рекламных вставок.

7.4 Использование web-аналитики

На современных предприятиях, имеющих представительство в Интернете, уделяется большое внимание эффективному функционированию созданных сайтов, разработка которых требует вложения значительных средств. Анализ информации, характеризующей работу сайта, позволяет путем улучшения его функционирования повысить и эффективность бизнеса.

Инструменты, позволяющие получить такую информацию - это инструмены web-аналитики. Наиболее популярными и востребованными инструментами web-аналитики являются пакеты Google Analytics, Яндекс Метрика, Webtrekk и другие. Аналитические пакеты используются не только в сайтах предприятий, но и в социальных сетях (например, ВКонтакте) или публичных сервисах (YouTube).

Web-аналитика появилась после длительного периода использования простых счетчиков посещений сайтов, постепенно преобразовавшихся в статистические пакеты, которые генерировали большое количество данных, относящихся к работе сайта, но только аналитические пакеты прврати эти данные в полезную информацию.

Пакеты web-аналитики генерируют сотни различных показателей и отчетов, интерпретация которых и составляет профессиональную задачу web-аналитика.

Умение грамотно использовать полученную информацию позволяет web-аналитикам изменять взаимодействие с клиентами таким образом, чтобы увеличивать коэффициент перехода (отношение числа клиентов, сделавших покупку или купивших услугу к общему числу посетителей сайта) и тем самым решить поставленную перед сайтом и предприятием бизнес-задачу.

Показатели, полученные с помощью аналитических пакетов, могут быть ориентированы в соответствии с целями предприятия и сайта. Так, возможность сегментировать данные о клиентах и их поведении дает более глубокое понимание их взаимодействия с web-сайтом, что может быть использовано для увеличения эффективности сайта и, соответственно, увеличению эффективности бизнеса.

Показатели, которые генерируются аналитическими пакетами, можно условно разделить на три группы:

-анализ посетителей;

-источники трафика;

-анализ контента.

Анализ посетителей позволяет выявить уникальных посетителей, новых и вернувшихся, а также осуществивших конверсию цели. Можно определить длительность пребывания клиента на сайте, а сделать правильный вывод о причине длительного или короткого пребывания посетителя возможно только при анализе дополнительных показателей или

их графическй интерпретации. К этой же группе относится глубина посещения (на сколько страниц вглубь сайта прошел пользователь) и показатель отказов (сколько клиентов покинули сайт и с какой страницы это произошло). Если пользователи покидают сайт с первой страницы, то встает вопрос, тех ли клиентов привлек сайт и откуда приходит их трафик.

Кроме того, интерес представляют технические характеристики компьютера пользователя – тип браузера, операционная система и сетевые характеристики.

Вторая группа показателей дает возможность определить, откуда пользователи пришли на сайт. «Прямой» трафик обеспечивается при наборе соответствующего URL сайта в браузере и свидетельствует об известности сайта и, соответствнно, хорошей работе маркетологов.

Возможно, клиенты приходят на сайт в результате поиска в поисковых системах – это так называемый «органический» трафик. В этом случае следует отметить работу специалистов SEO (Search Engine Optimized).

Наконец, пользователи могут прийти на сайт, используя рекламные баннеры на сайтах-реффererах. Это свидетельствует об более или менее удачной рекламной кампании.

В третьей группе показателей производится анализ страниц входа и выхода посетителей, определяется направление дальнейшего движения, а также проверяется удобство навигации и правильность ее функционирования.

Кроме данных, полученных с помощью отчетов соответствующих пакетов, web-аналитика ипользует множество методик, позволяющих собрать качественные данные о клиентах, например, лабораторная проверка применимости, при которой в лабораторных условиях при наблюдении разработчиков и менеджеров фокусная группа пользователей решает поставленные перед ними задачи, используя проверяемый сайт, опрос потенциальных клиентов и многое другое.

Web-аналитика – это объективное отслеживане, сбор, измерение и анализ количественных данных Интернета с целью увеличения эффективности web-сайтов и инициатив web-маркетинга. Web-аналитика является достаточно молодой, но быстро развивающейся и востребованной областью знаний.

7.5 Об авторском праве

Общество исходит из базовой концепции, в соответствии с которой, люди могут сосуществовать, если индивидуальные права защищаются законами или правилами. Эта концепция является основой цивилизации. В настоящее время такие законы особенно значимы для тех, кто использует и создает информацию в Интернет. Однако в большинстве случаев, в отношении электронной информации эти законы весьма неясны.

Если кто-то создает в Сети графические изображения, текст или размещает видео клип, то он и обладает авторским правом на свои создания независимо от того размещено ли уведомление об этом на web-страницах.

Это не означает, что нельзя использовать работы, защищенные авторским правом. Например, если процитировать несколько строк из статьи и указать источник и автора, то такие действия не противоречат законодательству. Можно получить письменное разрешение на использование работ, подпадающих под закон об авторских правах. Для этого нужно связаться с автором, издателем или другим лицом, имеющим лицензию на интересующий вас материал. В полной мере это касается и аудио информации.

Еще одной возможностью использовать информацию с чужих сайтов является размещение ссылки на интересующий вас объект. Однако подобное действие не является полностью оправданным (а является, скорее, неэтичным), так как автор, возможно, не предполагал такое использование своего материала.

В Интернет существует множество сайтов, материалы которых авторы предназначают для свободного использования, в частности, это многие личные сайты. Коммерческие фирмы, напротив, очень жестко относятся к защите своих логотипов, товарных знаков, пиктограмм и преследуют нарушителей своих прав.

Однозначное толкование законов, связанных с Интернет, затруднено благодаря "всемирному" характеру Сети. Законы, контролирующие информацию в одной стране, не обязательно таковы в других странах мира. Правовые вопросы в киберпространстве – очень важная тема, с которой должен познакомиться любой человек, желающий поместить собственную информацию в Интернет.

Литература

1. Хоган, Брайан. HTML5 и CSS3. Веб-разработка по стандартам нового поколения.-СПб.: Питер, 2012.-272с.: ил.

2. Пилигрим, Марк. Погружение в HTML5: перев. с англ. – СПб.: БХВ-Петербург, 2011.- 304 с.: ил.

3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. Руководство.- Питер, 2016.-768 с.:ил.

4. Мейер, Эрик. CSS – каскадные таблицы стилей. Подробное руководство, 3-е издание.- Символ-Плюс, 2008.- 576 с.:ил.

5. Евсеев, Д.А. Web-дизайн в примерах и задачах. Учебное пособие.- КноРус, 2016.-264 с.:ил.

6. Фримен, Э. Изучаем программирование на JavaScript.- Питер, 2015.- 640 с.:ил.

7. Дунаев, В.В. HTML, скрипты и стили, 3-е изд. [Электронный ресурс] / В. В. Дунаев. - СПб. : БХВ-Петербург, 2011. - 816 с.

8. Савельев, А. О. HTML 5. Основы клиентской разработки [Электронный ресурс] : учебное пособие /А. О. Савельев , А. А. Алексеев. – М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2012. - 166 с.

9. Стригина, Е. В. Web-девелопмент и web-дизайн [Электронный ресурс] : методические указания к выполнению курсовой работы / Е. В. Стригина ; рец. А. Д. Сотников ; Федеральное агентство связи, Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования "Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича". - СПб. : СПбГУТ, 2013.- 8 с.

10. Бердышев, С. Н. Искусство оформления сайта [Электронный ресурс] : практическое пособие /С. Н. Бердышев. – М. : Дашков и К, Ай Пи Эр Медиа, 2012. - 101 с.

11. Веселкова, Т. В. Эффективная эксплуатация сайта [Электронный ресурс] : практическое пособие / Т. В. Веселкова. – М. : Дашков и К, 2011. – 176 с.

12. Комолова, Н. HTML. Самоучитель. 2-е изд. [Электронный ресурс] / Н. Комолова, Е. Яковлева. - СПб. : Питер, 2011. - 288 с.

13. Электронный ресурс http://kirsanov.com/web.design/0.html

14. Электронный ресурс https://www.scribd.com/doc/61881197/Веб-дизайн-Книга-Якоба-Нильсена

15. Электронный ресурс http://www.htmlbook.ru

Оглавление