МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

С.А. Филиппов

Основы современного веб-программирования

Рекомендовано УМО «Ядерные физика и технологии»
в качестве учебного пособия
ля студентов высших учебных заведений

Москва 2011

УДК 004.738(073)
ББК 32.973.202я7
Ф 53

Филиппов С.А. **Основы современного веб-программирования**: Учебное пособие. М.: НИЯУ МИФИ, 2011. – 160 с.

В пособии представлены наиболее популярные технологии создания веб-систем: HTML, CSS, JavaScript, SSI, PHP, MySQL, а также обзор наиболее распространенных свободно-распространяемых и коммерческих веб-систем.

Предназначено для студентов-старшекурсников, студентов, проходящих обучение по сокращенной форме и получающих второе высшее образование.

Будет интересно преподавателям, которые планируют создание образовательных ресурсов в электронном виде.

Подготовлено в рамках Программы создания и развития НИЯУ МИФИ.

Рецензент д-р техн. наук, проф. А.И. Гусева

ISBN 978-5-7262-1402-3

Редактор Е.Г.Станкевич

Подписано в печать 15.12.2010. Формат 60×84 1/16.
Печ. л. 10,0. Уч.-изд. л. 14,0. Тираж 100 экз.
Изд. № 1/4/12. Заказ № 20

Национальный исследовательский ядерный университет «МИФИ».
115409, Москва, Каширское ш., 31

ООО «Полиграфический комплекс «Курчатовский».
144000, Московская область, г. Электросталь, ул. Красная, д. 42

Содержание

Предисловие

Сегодня можно уверенно констатировать, что Россия вступает в эпоху информационного общества. Основой, «кирпичами» этого общества являются веб-системы. Они определяют его возможности, влияют на решения, способствуют улучшению уровня жизни. В такой ситуации особенно востребованными становится знание основных механизмов создания и функционирования подобных систем, умение правильно поставить задачу на ее разработку или же реализовать ее самостоятельно.

Данное учебное пособие посвящено рассмотрению основных технологий и принципов создания и обеспечения доступности конкурентоспособных веб-систем, среди которых:

- размещение систем в Интернете;
- формирование статических и динамических веб-страниц;
- готовые веб-системы.

В пособии приведены только основные, часто используемые технологии и их составные части. Для получения более обширной информации рекомендуется ознакомиться со списком рекомендуемой литературы.

Автор советует не только использовать многочисленные примеры из настоящей книги, но и смело экспериментировать, создавая собственные системы.

Раздел 1. Технологическая база веб-систем

1.1. Интернет

Интернет (англ. Internet, сокр. от Interconnected Networks – объединённые сети) – глобальная телекоммуникационная сеть информационных и вычислительных ресурсов. Служит физической основой для Всемирной паутины. Часто упоминается как Всемирная сеть, Глобальная сеть, либо просто Сеть. Представляет собой хаотичное объединение автономных систем, что не гарантирует качества связи, но обеспечивает хорошую устойчивость и независимость функционирования системы в целом от работоспособности какого-либо ее участка.

Когда слово internet написано со строчной буквы, оно означает просто объединение сетей (англ. interconnected networks) посредством маршрутизации пакетов данных. В этом случае не имеется в виду глобальное информационное пространство Интернет (англ. Internet). В неанглоязычной или нетехнической среде эти понятия обычно не различают.

Словарь русского языка Российской академии наук под редакцией В. В. Лопатина рекомендует написание слова с прописной буквы: Интернет (род. падеж – Интернета). Написание со строчной буквы используется в сложных словах, таких как «интернет-портал» и «интернет-магазин».

Некоторые издания (Яндекс, «Коммерсантъ», «Наука и жизнь» и др.) считают, что собственное имя Всемирной сети уже стало нарицательным и пишут «интернет» с маленькой буквы.

Слово «Интернет» склоняется по правилам русской грамматики как существительное мужского рода, ничем не отличаясь от таких слов, как интернат и интерфейс. Поэтому писать следует: «в Интернете», «структура Интернета».

После запуска Советским Союзом искусственного спутника Земли в 1957 году министерство обороны США посчитало, что на случай войны Америке нужна надёжная система передачи информации. Агентство передовых оборонных исследовательских проектов США (DARPA) предложило разработать для этого компьютерную сеть. Разработка такой сети была поручена Калифорнийскому

университету в Лос-Анджелесе, Стэнфордскому исследовательскому центру, Университету штата Юта и Университету штата Калифорния в Санта-Барбаре. Компьютерная сеть была названа ARPANET (англ. Advanced Research Projects Agency Network) и в 1969 году в рамках проекта сеть объединила четыре указанных научных учреждения. Все работы финансировались министерством обороны США. Затем сеть ARPANET начала активно расти и развиваться, её начали использовать учёные из разных областей науки.

К 1971 году были разработаны протокол передачи файлов FTP (file transfer protocol), а также первая программа для отправки электронной почты по сети. Эта программа сразу стала очень популярна.

В 1970-х годах сеть в основном использовалась для пересылки электронной почты, тогда же появились первые списки почтовой рассылки, новостные группы и доски объявлений. Однако в то время сеть ещё не могла легко взаимодействовать с другими сетями, построенными на других технических стандартах. К концу 1970-х годов начали бурно развиваться протоколы передачи данных, которые были стандартизированы в 1982–1983 годах. 1 января 1983 года сеть ARPANET перешла с протокола NCP на TCP/IP, который успешно применяется до сих пор для объединения (или, как ещё говорят, «наслоения») сетей. Именно в 1983 году термин «Интернет» закрепился за сетью ARPANET.

В 1984 году разработана система доменных имён (англ. Domain Name System, DNS), а в 1988-м – протокол Internet Relay Chat (IRC), благодаря чему в Интернете стало возможно общение в реальном времени (чат).

В 1989 году в Европе, в стенах Европейского совета по ядерным исследованиям (фр. Conseil Européen pour la Recherche Nucléaire, CERN), родилась концепция Всемирной паутины. Её предложил знаменитый британский учёный Тим Бернерс-Ли, он же в течение двух лет разработал протокол HTTP, язык HTML и идентификаторы URI.

В 1991 году Всемирная паутина стала общедоступна в Интернете, а в 1993-м появился знаменитый веб-браузер NCSA Mosaic. Сеть набирала популярность.

В 1995 году Всемирная паутина стала основным поставщиком информации в Интернете, обогнав по трафику протокол пересылки

файлов FTP. Был образован Консорциум всемирной паутины (W3C). Можно сказать, что Всемирная паутина преобразила Интернет и создала его современный облик. С 1996 года Всемирная паутина почти полностью подменяет собой понятие «Интернет».

Рунет – русскоязычная часть всемирной сети Интернет. Более узкое определение гласит, что Рунет – это часть Всемирной паутины, принадлежащая к национальным доменам .su, .ru и .рф. 1987–1994 годы стали ключевыми в зарождении русскоязычного Интернета. 28 августа 1990 года профессиональная научная сеть, выросшая в недрах Института атомной энергии им. И. В. Курчатова и ИПК Минавтопрома и объединившая учёных-физиков и программистов, соединилась с мировой сетью Интернет, положив начало современным российским сетям. 19 сентября 1990 года был зарегистрирован домен первого уровня .su в базе данных Международного информационного центра InterNIC. В результате этого Советский Союз стал доступен через Интернет. 7 апреля 1994 года в InterNIC был зарегистрирован российский домен .ru. А 12 мая 2010 года домен .рф

1.2. Хостинг

Для того чтобы создаваемая система была доступна в Интернет, требуется решить следующие задачи:

1) выбрать вариант хостинга, т.е. то, каким образом будет организована постоянная доступность ваших материалов в Интернете;
2) получить прописку в Интернете (постоянный IP-адрес, адрес домена);
3) загрузить материалы на сервер и открыть доступ к ним.

Рассмотрим каждый пункт более подробно.

Хостинг

Хостинг (англ. hosting) – услуга по предоставлению вычислительных мощностей для физического размещения информации на сервере, постоянно находящемся в сети (обычно Интернет). Хостингом также называется услуга по размещению оборудования клиента на территории провайдера с обеспечением подключения

его к каналам связи с высокой пропускной способностью (колокация, от англ. colocation).

Обычно под понятием «услуги хостинга» подразумевают как минимум услугу размещения файлов веб-системы на сервере, на котором запущено ПО, необходимое для обработки запросов к этим файлам (веб-сервер). Как правило, в услугу хостинга уже входит предоставление места для почтовой корреспонденции, баз данных, DNS, файлового хранилища и т.п., а также поддержка функционирования соответствующих сервисов.

Хостинг баз данных, размещение файлов, хостинг электронной почты, услуги DNS могут предоставляться отдельно как самостоятельная услуга либо входить в понятие «услуги».

По условиям предоставления хостинг часто разделяется на платный и бесплатный. Обычно компания, предоставляющая бесплатный хостинг, зарабатывает путем показа рекламы на страницах, размещенных на нем. Бесплатный хостинг, как правило, медленнее платного, предоставляет только базовые услуги и потенциально менее надёжен.

Технологически, что в первую очередь сказывается на стоимости, хостинг может быть организован следующим образом:

- виртуальный хостинг, когда на одном сервере (компьютере) размещено много веб-сайтов разных заказчиков. Небольшие дешевые хостинг-провайдеры часто пренебрегают безопасностью и вообще не разграничивают привилегий пользователей, что позволяет одному пользователю на сервере иметь доступ к сайтам сотен других пользователей. У более крупных и дорогих провайдеров эта проблема, как правило, решена;
- виртуальный частный (выделенный) сервер (VPS или VDS), когда предоставляется место на диске, часть общей памяти, процессорное время сервера. Выглядит для пользователя так же, как и выделенный сервер, но физически на одном реальном сервере располагается несколько виртуальных серверов. Услуга предназначена для проектов средней тяжести. В связи с тем, что четко разделить все ресурсы сервера невозможно (в частности, операции ввода/вывода, ресурсы сетевой карты и др.), а многие провайдеры VPS продают ресурсов боль-

ше, чем есть на сервере, надеясь на то, что клиент задействует выделенный ему потенциал не полностью (оверселлинг), часто заявленная мощность VPS-сервера не соответствует реальной;

- выделенный сервер, когда сервер (компьютер) предоставляется целиком. Используется для реализации нестандартных задач (сервисов), а также размещения «тяжёлых» веб-проектов, которые не могут сосуществовать на одном сервере с другими проектами и требуют под себя все ресурсы сервера;
- колокация (collocation), когда предоставляется место в дата-центре провайдера для оборудования клиента (обычно путем монтажа в стойке) и подключение его к интернету;
- облачные хранилища, когда предоставляется возможность арендовать необходимый единый машинный ресурс, который может обеспечиваться физически несколькими серверами (компьютерами);
- облачные вычисления, когда арендуются не аппаратные ресурсы, а услуга (сервис редактирования документов, сервис автоматизации бухгалтерии и т.п. согласно принципам SaaS, WaaS, DaaS, IaaS, PaaS, HaaS и т.д.). Как правило на таких хостингах размещаются общие ресурсы и виртуальные офисы.

Один из важных критериев при выборе варианта с использованием оборудования хостинговой компании – используемая операционная система, поскольку от этого часто зависит программное обеспечение, которое может быть использовано для реализации поставленных целей. Важным аспектом описания хостинга является наличие таких служб и возможностей, как поддержка:

- CGI: Perl, PHP, Python, ASP, Ruby;
- .htaccess/.htpasswd (для веб-сервера Apache);
- баз данных.

Хостинг как услугу сравнивают и описывают

- по количественным ограничениям, включающим в себя:
  - размер дискового пространства,
  - количество месячного трафика,

- количество сайтов, которые можно разместить в рамках одной учетной записи,
- количество FTP-пользователей,
- количество ящиков электронной почты и объём места, предназначенного для них,
- количество баз данных и места под базы данных,
- количество одновременных процессов на пользователя,
- количество ОЗУ и максимальное время исполнения, выделяемое каждому процессу пользователя;

- и такие качественные ограничения, как:
  - свободные ресурсы процессора, оперативной памяти, влияющие на быстродействие сервера;
  - пропускная способность каналов, влияющая на загрузку информации;
  - удаленность оборудования хостинговой компании от целевой аудитории сайта, влияющая на скорость загрузки информации.

Некоторые хостинговые компании предоставляют бесплатный тест на определённый период, по истечении которого пользователь должен определиться, подходит ли для него выбранная хостинговая компания и имеет ли смысл оплачивать большие периоды. Как правило, такие тесты предоставляются только владельцам доменов второго уровня во избежание спекуляций с тестовыми аккаунтами.

Интернет-адресация

После выбора плана хостинга необходимо, чтобы к серверу можно было получить доступ через Интернет. Если это ваш собственный или выделенный только под вас сервер, то хостинговая компания выдает ему постоянный IP-адрес вида 72.16.21.34, по которому можно получить доступ к серверу по стандартным интернет-протоколам через установленные на нем службы (ftp, ssh, http и т.д.).

Второй шаг (или первый в случае аренды виртуального сервера) – приобретение доменного имени, представляющего собой уни-

кальную человекочитаемую буквенно-цифровую последовательность, позволяющего не только поставить в соответствие IP-адресу имя (на один IP-адрес может ссылаться несколько доменных имен), но и определить, какие именно материалы с сервера будут доступны пользователям.

Разделяют домены общего пользования:

- .com (commercial) – для коммерческих организаций;
- .org (organizations) – для некоммерческих организаций;
- .net (networks) – для организации представительств сетей/провайдеров;
- .biz (business organizations) – для любых организаций, введен после того, как в доменах .com и .org закончились простые словосочетания;
- .info (information) – домен для любых нужд;
- .mobi – для сайтов, ориентированных на работу с мобильными устройствами;
- .tel – для хранения и управления персональными и корпоративными контактными данными;
- .name – для частных лиц.

И национальные двухбуквенные домены согласно международному стандарту ISO 3166-1:

- .ru – Россия;
- .su – СССР (ныне Россия);
- .us – США;
- .de – Германия;
- .fr – Франция и т.д.

Необходимо помнить, что для эффективного использования доменное имя должно удовлетворять ряду условий:

- должно легко запоминаться;
- быть достаточно коротким;
- быть простым в написании, во избежание ошибок пользователей при его наборе;
- быть легко произносимым;
- содержать название компании либо обозначать сферу ее деятельности, основной продукт или услугу и т.д.

В зоне .ru присутствуют следующие ограничения на регистрацию:

- доменное имя должно состоять по меньшей мере из двух символов и не превышать 64 символов;
- доменное имя не может начинаться или заканчиваться дефисом;
- доменное имя не содержит двух дефисов подряд;
- содержащие слова, противоречащие общественным интересам, принципам гуманности и морали (в частности, слова непристойного содержания, призывы антигуманного характера, оскорбляющие человеческое достоинство либо религиозные чувства);
- входящие в перечень зарезервированных доменных имен.

Преобразованием доменного имени в IP-адрес осуществляют DNS-серверы (поддерживающие протокол Domain Name System). Обычно используются собственные DNS-серверы или серверы хостинговой компании, но могут использоваться и любые другие, с владельцами которых удастся договориться о внесении соответствующих записей о преобразовании. Обязательным условием является то, что должно быть определено минимум 2 DNS-сервера: primary (основной) и secondary (вторичный). Это сделано для обеспечения отказоусточивости преобразования: если один из серверов не работает, то второй обеспечивает необходимую обработку DNS-запросов пользователей. В связи с этим есть еще одно требование: DNS-серверы должны находиться в разных сетях подкласса С, т.е. первые два числа IP-адреса могут совпадать, а третье – всегда отличается (123.123.x.16).

Перед регистрацией домена соответствующие DNS-записи должны быть размещены на DNS-серверах.

Лидер среди регистраторов доменов в России – Региональный Сетевой Информационный Центр (http://www.nic.ru), который предоставляет удобный сервис по регистрации и оплате доменов в три шага:

1) заключить договор (можно в безписьменном виде);
2) внести авансовый платеж;

3) заказать необходимый набор доменов, в том числе указать соответствие имени домена и IP-адреса для размещения этой информации в корневом DNS-сервере регистратора.

Если все шаги пройдены успешно, то регистратор списывает средства с вашего счета и приступает к тестированию DNS-записей на ваших серверах. При получении положительного отклика (запись существует) запись фиксируется в корневом DNS-сервер регистратора, тиражируется среди глобальных корневых DNS-серверов (всего их 13) и кэшируется DNS-серверами крупных провайдеров. Данная процедура занимает от 6 часов до суток, после чего любой пользователь Интернет сможет получить доступ к вашему серверу по доменному имени.

К моменту, когда домен будет зарегистрирован, желательно подготовить сервер к эксплуатации: установить необходимые серверные службы (веб-сервер, ftp-сервер, почтовый сервер и т.п.) и наполнить данные системы содержанием, чему и посвящена основная часть данного пособия.

Домен можно приобрести заранее. В этом случае регистратор позволяет зарезервировать имя домена, не внося запись в корневой DNS-сервер регистратора.

1.3. Классификация веб-технологий

Все технологии, применяемые при создании веб-систем, делятся на два основных класса: исполняемые на клиенте средствами обозревателя Интернет (HTML, CSS, JavaScript, Flash, ActiveX и т.п.) и исполняемые на сервере средствами веб-сервера (SSI, PHP, ASP, Perl, Python и т.п.) и связанных с ним систем (MySQL, PostgreSQL, MSSQL и т.п.).

Приложения, исполняемые на сервере, практически ничем не ограничены по сложности: могут выполнять любые преобразования информации и затем формировать поток данных, который может быть визуализирован пользователю обозревателем Интернет.

Соответственно форматы данных, которые могут быть обработаны на клиенте, ограничены достаточно узким набором технологий, стандартов и определенными рамками, что позволяет унифицировать рабочее место пользователя и не требовать от него уста-

новки какого-либо дополнительного программного обеспечения кроме обозревателя Интернет.

Рис. 1.1. Взаимодействие серверных и клиентских технологий

1.4. Инструментарий разработчика

Сегодня наиболее популярным серверным комплексом является так называемый LAMP – акроним, составленный из первых букв следующего программного обеспечения:

- Linux – операционная система;
- Apache – веб-сервер;
- MySQL – система управления базами данных;
- PHP – скриптовый язык программирования, используемый для создания веб-приложений.

Данный комплекс позволяет решить большинство из стоящих сегодня перед разработчиками веб-систем задач на приемлемом по скорости обработки информации уровне.

Локальные LAMP

Совершенно очевидно, что для создания веб-приложений некорректно использовать возможности боевого сервера, так как при возникновении ошибок, что неизбежно происходит при создании приложений, может возникнуть отказ от обслуживания пришедших на сервер клиентов, что в свою очередь может испортить репутацию компании. Для данных случаев, а также тогда, когда боевой сервер еще не настроен, существуют следующие локальные пакеты

разработки веб-приложений, содержащие всё необходимое серверное обеспечение, в том числе и операционную систему, в случае если разработку предполагается проводить в наиболее близких к боевому серверу условиях на виртуальной машине:

- «Денвер» – наиболее популярный в России комплекс, поскольку полностью русифицирован и не требует какой-либо установки на компьютер кроме как копирование файлов в определенную директорию. В самой простой версии содержит только AMP (http://www.denwer.ru). Существует только под ОС MS Windows (рекомендуется установить на свой компьютер для выполнения работ из разделов 5–7 настоящего пособия);
- XAMPP – международный кросс-платформенный пакет. Существуют версии под все основные операционные системы (http://www.apachefriends.org/ru/xampp.html);
- Bitnami – полноценные виртуальные машины, реализованные для плееров VMWare и VirtualBox (http://bitnami.org).

Данные пакеты позволяют обрабатывать всю серверную логику на локальной машине, а также подключать иные серверные приложения (Perl, Python, PostgreSQL и т.п.).

Алгоритм создания своего сайта в «Денвере»

Инсталлятор «Денвера» выполнен полностью на русском языке, в связи с чем установка не должна вызывать вопросов. При этом рекомендуется при запросе на создание виртуального диска выбрать вариант 2 (только при запуске «Денвера»), что позволит при необходимости удалить комплекс, стерев каталог, куда он установлен.

Управление «Денвером» сведено к следующим командам:

- старт сервера (файл denwer\Run.exe);
- остановка сервера (файл denwer\Stop.exe);
- перезапуск сервера (файл denwer\Restart.exe);
- остановка и отключение виртуального диска (файл denwer\SwitchOff.exe).

После установки и первого запуска комплекса рекомендуется сразу же запустить обозреватель Интернета, набрав в нем следующий адрес: **http://localhost**. После чего должна отразиться тестовая страница «Денвера». Выбрав те или иные ссылки, можно протестировать каждый компонент «Денвера» отдельно.

Возможные причины неработоспособности комплекса и их решения:

1) если есть файервол (или антивирус), то необходимо создать разрешающие правила для работы «Денвера» с портом 80 или отключить файервол
2) если установлена программа Skype, некоторые версии которой могут блокировать необходимые порты, то необходимо отключить ее и проверить работоспособность комплекса;
3) в некоторых случаях причиной проблемы может стать служба Internet Information Services(IIS). Чтобы отключить ее, проделайте следующий путь: Пуск → Панель управления → Установка и удаление программ → Установка компонентов Windows. В появившемся окошке убрать галочку с компонента Internet Information Services(IIS). Нажать на кнопку «Далее».

HTML-документы должны находиться в директориях /home/<имя_хоста>/www. В своих работах в рамках настоящего пособия рекомендуется в качестве имени хоста указывать свою фамилию (или имя) в латинской транскрипции. По умолчанию сконфигурированы три виртуальных хоста:

- http://localhost (содержит скрипты тестирования и различные утилиты);
- http://test1.ru;
- http://custom-host:8648 (хост, имеющий свой собственный IP-адрес и порт).

Поддерживаются также виртуальные хосты с доменными именами третьего и выше уровней. Примеры того, как Apache ищет директории документов, приведены в табл. 1.1.

Таблица 1.1
Принципы создания в Денвере доменов третьего и больших уровней

Доменное имя	Директория документов
abcd.test1.ru	/home/test1.ru/abcd
ab.cd.test1.ru	/home/test1.ru/ab.cd
test.localhost	/home/localhost/test
ab.cd.localhost	/home/localhost/ab.cd

Директория /usr/local содержит программные компоненты – выполняемые и конфигурационные файлы Apache, PHP, MySQL. Авторы комплекса постарались сохранить это расположение приближенным к принятому в операционной среде Linux. Но имеются серьезные отличия – не все компоненты пакета распределены по соответствующим директориям. Полной аналогии с ОС Linux в любом случае не реализовано, но данное размещение выполняемых и конфигурационных файлов позволяет несколько легче ориентироваться в компонентах сервера.

Редакторы скриптов

Переходя к процессу разработки веб-систем, необходимо уяснить, что, в отличие от существующих языков программирования, большинство веб-ориентированных технологий не требует какой-либо компиляции, т.е. создания бинарных исполняемых файлов (exe). Необходимую интерпретацию написанных кодов сначала выполняет серверное программное, а затем – обозреватель Интернет. Чтобы писать для Интернета, достаточно иметь простейший текстовый редактор на подобии Notepad или vi. Однако такой подход не всегда приемлем, так как чреват большим количеством ошибок. Данный пробел устраняет большое количество редакторов как платных, так и бесплатных, позволяющих подсвечивать синтаксические конструкции, присущие тем или иным веб-языкам:

- ActiveState Komodo Edit, доступный для большинства операционных систем, поддерживающий все основные стандарты создания веб-приложений, бесплатный (https://www.activestate.com/komodo-edit/).

- Aptana Studio, доступный для большинства операционных систем, бесплатный, ориентированный на стандарты на стороне клиента (http://www.aptana.org/studio).
- Adobe DreamWeaver, наиболее распространенный в России платный редактор, поддерживающий на сегодня все существующие стандарты и имеющий большое количество дополнительных модулей, автоматизирующих разработку веб-приложений. Доступен только для MS Windows.

1.5. Проблемы несовместимости

Сегодня проблема нестандартной визуализации HTML-документов уходит в прошлое, но до тех пор, пока у пользователей установлены старые обозреватели Интернет, разработчики веб-систем сталкиваются с необходимостью включать в свои коды дополнительные проверки и функции, позволяющие интернет-сайту выглядеть более-менее одинаково в разных обозревателях Интернет. Данную проблему стали называть «войной обозревателей Интернет», в основе которой – несовершенство международных стандартов и необходимость изобретать собственные решения для удовлетворения потребностей пользователей.

Наибольшие различия возникли в поддержке JavaScript – языка сценариев, придающего интерактивность документам, и CSS – каскадных таблиц стилей, определяющих внешний вид HTML-документа. В результате многие разработчики либо «оптимизировали» свои сайты для конкретного обозревателя (особенно это было удобно, когда доля Microsoft Internet Explorer достигала 99%), которые практически не читались в других обозревателях из-за ужасной визуализации, либо создавали их с учетом технологий, которые одинаково визуализировались во всех обозревателях, что зачастую лишало их привлекательного оформления.

Особенно пренебрегающим международными интернет-стандартами, утвержденными Консорциумом Всемирной паутины (англ. World Wide Web Consortium, W3C), считаются разработчики Microsoft Internet Explorer. Наиболее хорошо учитывают существующие стандарты Mozilla Firefox и Google Chrome.

Задачи для самоконтроля

1. В каком году разработана система доменных имен?
2. Найдите и составьте сравнительную таблицу по не менее чем десяти платным и бесплатным хостинговым компаниям.
3. Найдите не менее пяти регистраторов доменных имен и сопоставьте предлагаемые ими условия, в том числе дополнительные услуги.
4. Какие технологии применяются в Microsoft Internet Explorer, но не применяются в других обозревателях Интернет?

Раздел 2. Гипертекстовый язык разметки (HTML)

2.1. Основные определения

HTML (от англ. HyperText Markup Language – «язык разметки гипертекста») – стандартный язык логической разметки документов в Интернете. HTML – не инструкции по визуализации текста (это задача CSS, см. раздел 3) и не язык программирования (это задача JavaScript и PHP, см. разделы 4 и 5). Он предназначен исключительно для структурирования текста. HTML – инструмент, позволяющий сказать: этот текст является абзацем, этот – заголовком, этот – цитатой, этот – ссылкой, этот – списком, а этот объект – изображение и т.д. Такой подход позволяет производить машинную обработку текстов и, при необходимости, формировать простейший визуальный ряд.

Программные средства, которые на основе инструкций HTML, помещенных в обычный текстовый файл, осуществляют интерпретацию текста и графики, получили название browser (он же – обозреватель Интернет). При этом по умолчанию обозреватели Интернет визуализируют инструкции в соответствии с общепринятыми нормами (далее – Стандартное отображение).

Рекомендуется все HTML-документы создавать с расширением html, чтобы упростить задачу и серверному и клиентскому программному обеспечению в идентификации содержания файла.

Любой HTML-документ – это набор элементов вида

```
"элемент" := <"имя элемента" ["список атрибутов">
содержание элемента <]/["имя элемента"]>
```

Конструкции, выделенные в треугольные скобки, называются тегами. В случае наличия содержания элемента выделяют открывающий тег вида **<"имя элемента" ["список атрибутов"]>** и закрывающий тег вида **</"имя элемента">**.

Пример элемента:

```
<p>текст параграфа</p>
```

Имена элементов могут набираться в любом регистре, т.е. ****, **<STRONG>** и **<sTRong>** равнозначны. Также обозрева-

тель никак не обрабатывает символы табуляции и пробела, если их больше одного, т.е. если в тексте ставится несколько пробелов или символов табуляции, то на экран выводится только один пробел в обоих случаях.

Элементы должны либо следовать друг за другом, либо быть вложены один в другой. Например:

```
<p><em>Выделяемый текст</em> другой текст</p>
<p>второй абзац</p>
```

Атрибуты позволяют управлять свойствами элемента, всегда включаются в открывающий тег элемента и имеют вид

```
имя_атрибута="значение_атрибута"
```

Друг от друга и от имени элемента атрибуты отделяются пробелами. Например, элемент **img** (изображение) имеет атрибут **src**, указывающий расположение файла с изображением, и атрибут **alt**, задающий альтернативный текст на тот случай, если обозреватель Интернет не отображает графику и для вывода всплывающей подсказки к изображению

```
<img src="mephi.gif" alt="МИФИ" />
```

Необходимо обратить внимание, что тег **img**, как и некоторые другие элементы, не требует закрывающего тега, так как не имеет содержания. В этом случае можно, ставить перед закрывающей треугольной скобкой символ дроби «/». Во всех остальных случаях закрывающий тег **обязателен**.

2.2. Структура HTML-документа

Как уже было сказано ранее, HTML-документ представляет из себя простой текстовый файл. Указанный файл, если он не включает в себя серверные сценарии, должен иметь расширение html и следующую общую структуру:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
```

```
<!-- Заголовок страницы и техническая информация -->
</head>
<body>
<!-- Текст, изображения, таблицы -->
</body>
</html>
```

Элемент **<!DOCTYPE>** предназначен для указания типа текущего документа – DTD (document type definition, описание типа документа). Он необходим, чтобы обозреватель понимал, как следует интерпретировать текущую веб-страницу, ведь HTML существует в нескольких версиях. Кроме того, существует XHTML (EXtensible HyperText Markup Language, расширенный язык разметки гипертекста), похожий на HTML, но различающийся с ним по синтаксису.

Тег **<html>** указывает, где начинается и заканчивается HTML-документ.

Элемент **<!-- код -->** позволяет вставлять в HTML-документ комментарии или комментировать код, который никогда не будет визуализирован пользователю.

Заголовок HTML-документа

Тег **<head>** определяет начало и конец заголовка документа, где описываются технические параметры документа: правила отображения, ключевые слова, подключение дополнительных программных модулей. Теги и тексты, находящиеся в этом разделе, не отображаются на веб-странице.

Тег **<title>** используется для отображения строки текста в левом верхнем углу окна обозревателя, а также для трансляции названия сайта и краткого содержания страницы. Текст из данного тега применяется поисковыми системами для формирования ссылки на страницу, что придает особую значимость содержимому тега с учетом того, что поисковые системы организуют до 90% посещений сайтов.

Тег **<meta>** используется для вставки данных, предназначенных для обозревателей и поисковых систем. Например, роботы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Хотя рассматриваемый тег всего один, он имеет множество атрибутов, позволяющих описывать разные данные. Наиболее важной для корректного воспроизведения всего HTML-документа является конфигурация тега, определяющая в какой кодировке написаны символы страницы.

При оформлении HTML-документов в одном из редакторов в ОС MS Windows необходимо вставлять тег

```
<meta http-equiv="content-type" content="text/html; char-
set=windows-1251">
```

При оформлении HTML-документов в одном из редакторов Linux необходимо вставлять следующий тег (при оформлении в двуязычных таблицах русский-английский):

```
<meta http-equiv="content-type" content="text/html;
charset=koi8-r">
```

либо тег (при применении стандарта кодирования символов, позволяющего представить знаки практически всех письменных языков)

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

В последнее время с кодировкой UTF-8 работают и некоторые редакторы ОС MS Windows.

Также в заголовке могут размещаться теги, ответственные за прикрепление к основному файлу каскадных таблиц стилей и клиентских сценариев (см. разделы 3 и 4).

Порядок размещения тегов в заголовке документа особого значения не имеет.

Тело HTML-документа

Тег **<body>** определяет расположение содержательной части HTML-документа. Теги, которые допустимо применять в теле HTML-документа, содержатся в пп. 2.3–2.9.

Большинство тегов, используемых в теле документа, подразделяются на блочные (block) и текстовые (inline) элементы. Блочные

элементы могут содержать как текстовые, так и другие блочные элементы. При отображении они всегда выводятся как отдельный абзац. Текстовые элементы могут содержать только текст и другие текстовые элементы, но не содержат блочных элементов. При отображении они выводятся в текущей строке.

Специальные символы

Символ в HTML может быть вызван как прямым образом, так и через его синоним, который выглядит как **&имя;** или **&#код;**

Такая возможность предусмотрена для использования в случае если символ отсутствует на клавиатуре, т.е. набрать его невозможно, либо для отделения описания HTML-элементов от содержания в целях корректной обработки HTML-инструкций, например в следующем случае:

```
<p>> 123</p>
```

Как мы сможем убедиться выше, в HTML используются четыре основных символа: открывающая треугольная скобка (<, имя – **<**), закрывающая треугольная скобка (>, имя – **>**), прямая кавычка (", имя – **"**) и амперсанд (&, имя – **&**).

В табл. 2.1 приведены другие часто употребимые символы.

Таблица 2.1

Имена часто используемых символов

&#167;	параграф	§	&#133;	многоточие	…	&#149;	жирная точка	•
&#169;	copyright	©	&#145;	одиночная открывающая кавычка	‘	&#150;	среднее тире (минус)	–
&#174;	registered	®	&#146;	одиночная закрывающая кавычка (апостроф)	’	&#151;	длинное тире	–
&#176;	знак градуса	°	&#132;	открывающая русская лапка	„		неразрывный пробел	
&laquo;	левая кавычка (левая елочка)	«	&#147;	закрывающая русская, открывающая английская лапка	“	&#8470;	знак номера	№
&raquo;	правая кавычка (правая елочка)	»	&#148;	закрывающая английская лапка	”	&#177;	плюс-минус	±

Экранная типографика

Правила экранной типографики нацелены на то, чтобы сделать текст максимально читаемым, профессиональным. Самое простое правило, регулирующее набор текстов, в том числе и для веб-страниц, можно сформулировать так:

Каждый знак текста должен функционально соответствовать своему предназначению, а не подменять собой другие и не подменяться другими.

Следствия из этого правила:

- есть тире (–), и его нельзя заменять дефисом (-);
- есть знак номера (№), и его нельзя подменять буквой N;
- есть русские кавычки («ёлочки»), и они не должны подменяться знаком дюйма, прямой кавычки ("). Существует старое правило, согласно которому внутри кавычек-ёлочек употребляются кавычки другого вида (кавычки „лапки“);
- есть буква «ё», и её совершенно незачем подменять буквой «е»;
- спецсимволы, например ©, не должны подменяться последовательностями символов вроде (c);
- апостроф ’ нельзя подменять знаком «одинарной кавычки» '.

Все остальные правила носят рекомендательный характер, но вытекают из здравого смысла. Например, тире в начале строки ассоциируется с прямой речью, поэтому нежелательно, чтобы тире внутри предложения оказывалось в начале строки. Этого легко избежать, привязав тире неразрывным пробелом к предыдущему слову.

Неразрывный пробел также удобен, чтобы привязывать предлоги и союзы к следующему за ними слову. Это особенно актуально для веб-страниц с большой шириной абзаца. Можно (во избежание появления висячих строк) привязывать последние короткие слова абзацев к предпоследним тем же неразрывным пробелом.

Следует оставлять на одной строке цифры телефонного номера для его целостного восприятия: для этого есть парный тэг **<nobr>**.

Пример текста:

…Когда И. И. Иванов увидел в газете (это была «Заморская правда» **№** 23) рубрику Weather Forecast®, он не поверил своим глазам – температуру обещали ±232° **C**.

Пример верстки:

…Когда **<nobr>**И. И. Иванов**</nobr>** увидел в** **газете (это была **«**Заморская правда**» № **23) рубрику Weather Forecast**®**, он** **не** **поверил своим глазам** —** температуру обещали**±**232**° **C.

2.3. Гиперссылки

Тег **<a>** можно охарактеризовать как главный тег Интернета. Благодаря ему Интернет получил свой сегодняшний вид, именно он организует взаимосвязь между документами и делает возможным создание гипертекста. Тег позволяет организовать ссылку с текста или изображения, помещенных между открывающим и закрывающим тегом, на любой Интернет-адрес, включая адреса электронной почты.

Вид: <a>… (текстовый элемент)
Индивидуальные атрибуты: href, target
Стандартное отображение содержания: синий подчеркнутый текст

Атрибут **href** определяет адрес, по которому обозреватель Интернет осуществляет переход при нажатии на ссылку.

Атрибут **target** указывает место, где будет открыта ссылка:

_self – в текущем окне (по умолчанию)
_blank – в новом окне

Пример создания ссылки:

```
<a href="news.html" target="_blank">Новости</a>
```

Для того чтобы создать ссылку на раздел документа (на закладку), необходимо после имени файла поставить знак # и название закладки:

```
<a href="news.html#today">Текущие новости</a>
```

Сами закладки определяются с помощью общего атрибута **id**, который может быть использован вместе с любым тегом и одновременно присваивающем тегу уникальный идентификационный код, для которого могут быть заданы свои таблицы стилей и (или) вызываться функции Javascript. Пример оформления закладки:

```
<p id="today">Текущие новости</p>
```

Ссылки могут быть абсолютными (обычно ставятся на внешние интернет-сайты) и относительными (ставятся только на документы собственного сайта с учетом файловой структуры размещения документов). Ссылка называется абсолютной, если путь к документу указывается полностью с названием протокола передачи документа и сайта:

```
<a href="http://www.mephi.ru/news.html">
```

Ссылка называется относительной, если она составлена с учетом месторасположения документа в файловой системе сервера либо определяется от корня сайта. Как известно, файловая система по сути имеет древовидную структуру (рис. 2.1.).

Таким образом, для построения относительной ссылки используют следующие принципы:

- Если файл находится в одном каталоге с документом из которого идет ссылка, то в качестве значения атрибута **href** пишется только имя файла. Так, например, выглядит ссылка из 4.html на 5.html:

```
<a href="5.html"> 5.html </a>
```

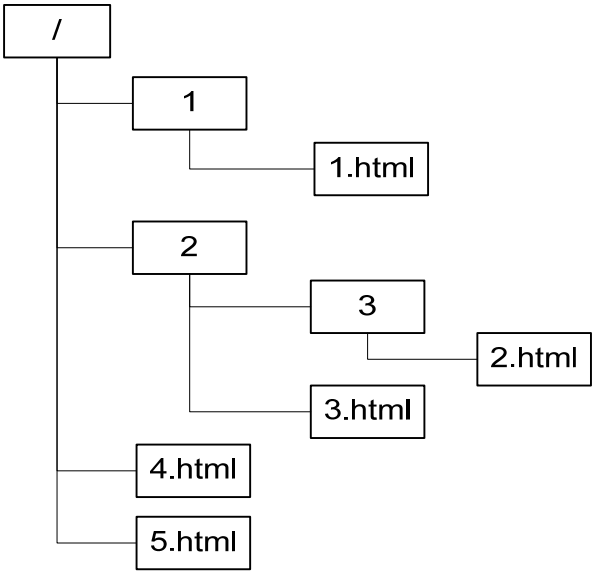


Рис. 2.1. Схема файловой системы

- Если документ находится в подкаталоге (подкаталогах), то по порядку перечисляются имена подкаталогов и затем пишется имя файла. Так, например, выглядит ссылка из 5.html на 2.html:

```
<a href="2/3/2.html"> 2.html </a>
```

- Если документ находится в надкаталогах, то для того, чтобы подняться на уровень выше, ставят две точки (..).Так, например, выглядит ссылка из 1.html на 4.html:

```
<a href="../4.html"> 4.html </a>
```

- Руководствуясь приведенными выше правилами, можно описать любую ссылку. Так, например, выглядит ссылка из 3.html на 1.html:

```
<a href="../1/1.html"> 1.html </a>
```

Для организации ссылки на адрес электронной почты в качестве протокола указывается mailto:

```
<a href="mailto:mephi@mephi.ru"> mephi@mephi.ru </a>
```

Из-за угрозы, исходящей от роботов, собирающих электронные адреса в Интернет, сегодня такую конструкцию впрямую практически не используют. Электронные адреса либо публикуют вообще без ссылки и с искажениями, доступными для интерпретации только человеку, например mephi[a]mephi.ru, либо используют алгоритмы динамического формирования целостного адреса электронной почты через функции Javascript, нацеленные на то, что из исходного кода страницы практически невозможно вычленить электронный адрес, а ссылка продолжает работать (см., например, http://www.tigir.com/hide_email.htm).

Правила организации ссылок верны для любых других атрибутов, если их значении подразумевает ссылку на документ или объект.

2.4. Фразовые элементы

Фразовые элементы определяют логические стили фрагментов текста.

Тег **** (emphasis) используется для выделения текста по аналогии с ударением для отдельной буквы, т.е. данный текст на порядок важнее, чем невыделенный:

Вид: …
Индивидуальные атрибуты: нет
Стандартное отображение содержания: курсив

Тег **** (strong emphasis) применяется для сильного выделения текста, т.е. данный текст на два порядка важнее, чем невыделенный:

Вид: …
Индивидуальные атрибуты: нет
Стандартное отображение содержания: полужирный

Тег **** логически отделяет текст от остального. Используется в основном в паре с каскадными таблицами стилей для изменения визуализации отдельных словосочетаний:

Вид: …
Индивидуальные атрибуты: нет
Стандартное отображение содержания: нет

2.5. Текстовые блоки

Тег ** ** (paragraph) определяет абзац текста:

Вид: …
Индивидуальные атрибуты: нет
Стандартное отображение содержания: новый абзац отделяется от предыдущего пустой строкой

Иногда в абзаце бывает необходимо в явном виде указать, что слово или предложение должны начинаться с новой строки для этого используется непарный тег **
**.

Тег **<pre>** (preformatted) предназначен для вывода текстов, в которых размещение слов в строках имеет существенное значение, например для стихов или фрагментов компьютерных программ:

Вид:

```
<pre>
   Люблю грозу в начале мая,
   Когда весенний, первый гром,
   Как бы резвяся и играя,
   Грохочет в небе голубом.
</pre>
```

Индивидуальные атрибуты: нет
Стандартное отображение содержания: перед текстом отображается пустая строка

Тег **<blockquote>** используется для обозначения длинных цитат:

Вид: <blockquote >…</blockquote >
Индивидуальные атрибуты: нет
Стандартное отображение содержания: перед текстом отображается пустая строка, сам блок сдвинут вправо

Теги с **<h1>** по **<h6>** (header) используются для создания заголовков в документе:

Вид: <h#>…</h# >
Индивидуальные атрибуты: нет
Стандартное отображение содержания: полужирным, от 1-го к 6-му уровню размер шрифта последовательно уменьшается

Тег ** ** – это блок в чистом виде, т.е. при стандартном отображении он никак не форматируется. Используется в основном в паре с каскадными таблицами стилей для визуализации тех или иных областей HTML-документа:

Вид: …
Индивидуальные атрибуты: нет
Стандартное отображение содержания: с новой строки

2.6. Мультимедийные объекты

Мультимедийные теги призваны решить вопрос включения в состав HTML-документа аудиовизуальных композиций: изображений, аудиофайлов, видеоклипов и их производных, управляющих элементов.

Тег **<img>** позволяет вставить изображение одного из стандартных Интернет-форматов (jpg, gif и png, все указанные форматы – растровые) в HTML-документ. Тег <img> непарный:

Вид: <img> (блочный элемент)

Индивидуальные атрибуты: src, alt, width, height
Стандартное отображение: изображение

Атрибут **src** определяет путь, по которому располагается изображение.

Атрибут **alt** позволяет задать подпись к изображению. Данная подпись отображается в том случае, если пользователь отключил отображение графики в обозревателе. Также подпись используется поисковыми системами для учета изображения в своей базе и воспроизведения при организации поиска изображений.

Атрибуты **width** и **height**, задающие размеры образа на экране по горизонтали и вертикали соответственно, позволяют обозревателю Интернет зарезервировать пространство для образа при загрузке документа и тем самым несколько ускорить отображение последнего. Также эти параметры позволяют провести масштабирование изображения, но этим лучше не пользоваться, так как в случае уменьшения реальных размеров изображения пользователь загружать больший объем данных, чем ему нужен, а в случае увеличения само изображение начинает выглядеть неприглядно.

Тег **<object>** позволяет вставить в HTML-документ любой объект, в том числе и изображения. Для отображения большинства объектов требуется установка дополнительных расширений для обозревателя Интернет:

Вид: <object>…</object> (блочный элемент)
Индивидуальные атрибуты: type, data, width, height, classid
Стандартное отображение: в соответствии с заложенными в объект свойствами

Атрибут **type** определяет тип данных объекта. Например, для объектов Adobe Flash это application/x-shockwave-flash

Атрибут **data** определяет путь, по которому располагается объект.

Атрибуты **width** и **height** задают размеры образа на экране по горизонтали и вертикали соответственно.

Атрибут **classid** точно задает, какое расширение обозревателя Интернет будет воспроизводить объект. Например, для объектов Adobe Flash это «clsid:D27CDB6E-AE6D-11cf-96B8-444553540000»

Непарный тег **<param>** применяется только в составе тега **<object>**. Указанный тег позволяет передавать объекту при его инициализации входные данные, которые могут быть командами, числовыми значениями и т.п., позволяющими влиять на визуализацию объекта. Поступающие данные объект обрабатывает самостоятельно без помощи обозревателя Интернет.

Вид: <param>
Индивидуальные атрибуты: name, value
Стандартное отображение: нет

Атрибут **name** задает название параметра, а атрибут **value** значение параметра.

Рассмотрим пример вставки Flash-объекта в HTML-документ:

```
<object type="application/x-shockwave-flash" da-
ta="http://mephi.ru/ban.swf" width="140" height="300">
        <param name="movie" value=" http://mephi.ru/ban.swf" />
</object>
```

В новом стандарте HTML5 ожидается поддержка тегов **<video>** и **<audio>**, названия которых говорят сами за себя.

2.7. Списки

HTML поддерживает три способа хранения и отображения списков. Любой список состоит из одного или нескольких элементов списков. Списки подразделяются на:

- маркированные (неупорядоченные) списки;
- нумерованные (упорядоченные) списки;
- списки определений.

Приведенный выше список называется маркированным и выглядит на языке HTML так:

```
<ul>
        <li>маркированные (неупорядоченные) списки;</li>
        <li>нумерованные (упорядоченные) списки;</li>
        <li>списки определений.</li>
</ul>
```

Нумерованный список выглядит аналогично, но его элементы нумеруются:

1. Первый элемент списка.
2. Второй элемент списка.

На языке HTML это выглядит так:

```
<ol>
        <li>Первый элемент списка.</li>
        <li>Второй элемент списка.</li>
</ol>
```

Тег **<ol>** имеет атрибут **start**, который позволяет определить, с какого числа производится нумерация:

```
<ol start="7"><li>…</li></ol>
```

Наконец, списки определений состоят из пар термин/определение, хотя их применение гораздо шире. Пример использования списка определений для составления театрального репертуара:

1 июля

А. К. Толстой. *Царь Федор Иоаннович*

9 июля

А. Островский. *Волки и овцы*

На языке HTML это записывается так:

```
<dl>
        <dt><strong>1 июля</strong></dt>
        <dd>А. К. Толстой. <em>Царь Федор
Иоаннович</em></dd>
        <dt><strong>9 июля</strong></dt>
        <dd>А. Островский. <em>Волки и овцы</em></dd>
</dl>
```

Списки могут вкладываться друг в друга, причем допускается вложение списков одного типа в списки другого типа. Рассмотрим пример, когда одно из определений имеет поясняющие материалы:

Определение структуры, содержания и методики преподавания объектно-ориентированного программирования базируется, прежде всего, на следующих аспектах:

1. Социальная потребность в наличии учебного курса, предназначенного для реализации предпрофильного обучения школьников.
2. Процесс обучения, построенный на формировании компетенции:
   - информационной,
   - коммуникативной.

На языке HTML приведенный фрагмент записывается так:

```
<dl>
        <dt>Определение структуры, содержания и методики преподавания объектно-ориентированного программирования базируется,  прежде всего, на следующих аспектах:</dt>
        <dd>
        <ol><li>Социальная потребность в наличии учебного курса, предназначенного для реализации предпрофильного обучения школьников.</li>
```

```
            <li>Процесс обучения, построенный на формировании
компетенции: <ul>
                        <li>информационной,</li>
                        <li>коммуникативной.</li>
                    </ul>
        </li></ol>
        </dd>
</dl>
```

2.8. Таблицы

Одним из наиболее мощных механизмов HTML является возможность представления данных в виде таблиц, т.е. возможность структуризации данных по строкам и столбцам. Образованные на пересечениях строк и столбцов ячейки могут содержать любые элементы HTML, в том числе и таблицы.

Любая таблица может иметь заголовок (элемент **caption**), содержащий ее краткое описание. Далее следует содержимое таблицы, которое может состоять из трех секций: необязательного элемента **thead**, необязательного элемента **tfoot** и одного или нескольких элементов **tbody**. Элементы **thead** и **tfoot** задают соответственно строки надзаголовка и подзаголовка таблицы, а элемент(ы) **tbody** – группы строк «тела» таблицы. Каждая группа состоит из строк, задаваемых элементами **tr**. В свою очередь каждый элемент **tr** – из элементов **th** или **td**, которые определяют содержимое ячеек заголовка и ячеек данных соответственно.

Парный тег **<table>** определяет начало и конец таблицы.

Вид: <table>…</table> (блочный элемент)
Индивидуальные атрибуты: cellspacing, cellpadding
Стандартное отображение: таблица без рамки, но с расстояниями между ячейками

Атрибут **cellspacing** задает расстояние между ячейками таблицы, а атрибут **cellpadding** – расстояние внутри ячеек (т. е. между содержимым ячейки и рамкой). Значения этих атрибутов всегда

применяются ко всем четырем сторонам ячейки. Чтобы не возникало проблем при применении каскадных таблиц стилей, оба этих атрибута устанавливают в 0.

Парный тег **<caption>** задает заголовок таблицы:

Вид: <caption>…</caption> (блочный элемент)
Индивидуальные атрибуты: align
Стандартное отображение: по центру и над таблицей

Атрибут **align** определяет способ вертикального выравнивания названия: top - помещает заголовок над таблицей (значение по умолчанию), bottom - помещает заголовок под таблицей.

Парный тег **<tr>** задает новую строку в таблице:

Вид: <tr>…</tr> (блочный элемент)
Индивидуальные атрибуты: нет
Стандартное отображение: нет

Парные теги **<th>** и **<td>** задают следующий столбец в таблице. Текст, размещенный между ними, ограничен строкой и столбцом, и в результате получается ячейка, куда и можно вводить необходимые данные. Тег **<th>** при этом еще информирует, что содержимое ячейки – это заголовок столбца или подстолбца:

Вид: <th>…</th> и <td>…</td> (блочный элемент)
Индивидуальные атрибуты: rowspan, colspan
Стандартное отображение: форматирование по левому краю, для th также добавляется выделение полужирным и форматирование от центра

Атрибуты **rowspan** и **colspan** задают соответственно количество строк и столбцов, занятых ячейкой. По умолчанию они равны 1. Специальное значение 0 указывает, что ячейка занимает все строки или столбцы до конца таблицы; однако это значение часто игнорируется обозревателями Интернет, и им лучше не пользоваться.

Пример таблицы (вместе с заголовком) приведен на рис. 2.2.

Табл.№. Информация по специальности 061800

Шифр	Специальность	Квалификация	Срок обучения
061800	Математические методы в экономике	Экономист-математик	Полный цикл обучения на дневном отделении составляет 5 лет
			Для выпускников специализированных средних учебных заведений – 3 года.
Наименование специальности действительно с сентября 2000 г.			

Рис. 2.2. Пример таблицы

Приведенная таблица в HTML-формате:

```
<table cellspacing="0" cellpadding="0">
        <caption>Табл. &#8470;. Информация по специальности
061800</caption>
        <thead>
        <tr>
            <th>Шифр</th>
            <th>Специальность</th>
            <th>Квалификация</th>
            <th>Срок обучения</th>
        </tr>
        </thead>
        <tbody>
        <tr>
            <td rowspan="2">061800</td>
            <td rowspan="2">Математические методы в экономи-
ке</td>
            <td rowspan="2">Экономист-математик</td>
            <td>Полный цикл обучения на дневном отделении со-
ставляет 5 лет</td>
        </tr>
        <tr>
```

```
            <td>Для выпускников специализированных средних
учебных заведений – 3 года.</td>
        </tr>
        </tbody>
        <tfoot>
        <tr>
            <td colspan="4"><em>Наименование специальности
действительно с сентября 2000 г. </em></td>
        </tr>
        </tfoot>

</table>
```

Так как в приведенной таблице только одно «тело», то теги **<thead>**, **<tbody>** и **<tfoot>** могут быть опущен.

2.9. Формы

Формы позволяют реализовать качественно новый уровень гиперсвязей между HTML-документами. Форма предоставляет пользователю возможность ввести требуемые данные и отправить их веб-серверу, который в свою очередь может их обработать и передать обозревателю не просто новый HTML-документ, но документ, сформированный в соответствии с введенными пользователем данными. Помимо этого формы позволяют сформировать электронное письмо, если в качестве ссылки стоит электронная почта. А также их можно использовать для исполнения клиентских сценариев (см. раздел 4 настоящего пособия), в частности проверки заполнения формы на полноту (см. Приложение 2).

Тег **<form>** позволяет определить начало и конец формы:

Вид: <form>…</form> (блочный элемент)
Индивидуальные атрибуты: action, method, name, onSubmit, onReset
Стандартное отображение: нет

Атрибут **action** указывает адрес HTML-документа, который будет загружаться после отправки данных на сервер. Обычно этот документ дополнен серверными сценариями (см. раздел 5), которые могут обработать и интерпретировать переданные данные. Если таких сценариев в документе нет, то переданные данные никак не обрабатываются.

Атрибут **method** позволяет определить метод, которым данные будут передаваться на сервер. Всего различают два метода:

1) **get** (принято по умолчанию). В этом случае обозреватель добавляет к адресу, указанному в **action**, знак вопроса '?' и набор данных формы, закодированный в соответствии с типом файла "application/x-www-form-urlencoded". Полученный адрес передается серверу по протоколу HTTP с пометкой "GET" (эта часть пользователю не видна). В этом случае допустимый объем передаваемых данных ограничен. Правила формирования адреса методом **get** вполне допустимо использовать и при формировании обычных гиперссылок, прописываемых в теге **<a>** (см. пример из раздела 5);
2) **post**. В этом случае обозреватель Интернет выполняет запрос HTTP с пометкой "POST" и отдельной командой HTTP передает данные формы. В этом случае при загрузке целевого HTML-документа пользователь никак не сможет определить, какие данные были переданы.

Атрибут **name** позволяет упростить обращение к форме из функций JavaScript, поскольку автоматически создает необходимый объект для операций.

Атрибут (событие) **onSubmit** позволяет перед отправкой формы вызвать какую-либо функцию JavaScript, и эта функция в том числе может прервать процесс отправки данных на сервер.

Атрибут (событие) **onReset** позволяет при сбросе данных формы к первоначальному виду (изначально определенному в HTML-документе) вызвать какую-либо функцию JavaScript.

В форму нельзя вкладывать другую форму (она не будет обрабатываться).

Тег **<input>** позволяет визуализировать управляющие элементы, с которыми может взаимодействовать пользователь и в которых фиксируются данные для отправки на сервер:

Вид: <input> (текстовый элемент)
Индивидуальные атрибуты: type, name, value, checked, readonly, size, maxlength, accept, onFocus, onBlur, onSelect, onChange
Стандартное отображение: в зависимости от значения атрибута type

В табл. 2.2. приведены значения, которые может принимать атрибут **type** и их описания.

Таблица 2.2

Значения атрибута type

№	Значение	Описание	Визуализация
1	text	Однострочное поле ввода текста (принято по умолчанию). Атрибут **value** задает начальное значение текста, **size** – размер поля ввода в символах, а **maxlength** – максимально возможное количество символов в данном поле. Атрибут **readonly** запрещает изменение содержание поля (применимо ко всем полям формы)	текст
2	password	Поле ввода пароля. Единственное отличие от поля ввода текста состоит в том, что вводимые символы маскируются, обычно заменой на звездочки. Однако, введенная информация передается серверу как обычный текст, поэтому этот элемент не обеспечивает уровня безопасности, достаточного для передачи номеров кредитных карт или другой существенно важной информации	********

Продолжение табл. 2.2

№	Значение	Описание	Визуализация
3	checkbox	Флаг. Атрибут **value** задает значение этого элемента, когда он выбран, а атрибут **checked** – выбран ли изначально (как изображено справа). Флаги образуют группу (передаются на сервер как массив данных), если у них одно и то же имя (атрибут **name**), заканчивающееся квадратными скобками: например, **flag[]**	☑ Флаг
4	radio	Переключатель. Атрибут **value** задает значение этого элемента, когда он выбран, а атрибут **checked** – выбран ли первоначально. Переключатели образуют группу (на сервер передается только значение выбранного элемента группы), если у них одно и то же имя (атрибут **name**). В этом случае при включении переключателя предыдущий переключатель переходит в отключенное состояние	◉ Переключатель
5	reset	Кнопка сброса формы. Необязательный атрибут **value** перекрывает текст кнопки, принятый по умолчанию. См. пример справа	Сброс
6	submit	Кнопка пересылки формы. Необязательный атрибут **value** перекрывает текст кнопки, принятый по умолчанию. Если форма содержит несколько таких кнопок, то атрибут **name** используется для передачи имени кнопки обработчику формы, что позволяет последнему осуществлять различные действия в зависимости от того, какая из кнопок пересылки формы была нажата	Отправить

Окончание табл. 2.2

№	Значение	Описание	Визуализация
7	button	Кнопка общего вида. Атрибут **value** задает текст кнопки, а атрибут **onClick** должен задавать сценарий, вызываемый при нажатии этой кнопки	
8	file	Селектор файлов. Атрибут **value** задает начальное имя файла, но обозреватели обычно игнорируют его из соображений безопасности. Необязательный атрибут **accept** задает список типов файлов, разделенных запятыми, которые поддерживаются сервером обработчика формы. Этот список может использоваться для отфильтровки только допустимых файлов, но современные обозреватели обычно игнорируют этот атрибут. Форма, содержащая селектор файлов, должна иметь атрибуты **method**="post" и **enctype**="multipart/form-data"	
9	hidden	Скрытый элемент. Никак не отображается, но позволяет сохранять дополнительные служебные данные, не требующие визуализации	

Атрибут **name** позволяет упростить обращение к управляющему элементу из функций JavaScript, так как автоматически создает необходимый объект для операций.

Атрибут (событие) **onFocus** позволяет вызвать какую-либо функцию JavaScript при получении элементом фокуса, т.е. когда пользователь предпринял шаги (нажатием мышки, с помощью табуляции) к тому, чтобы начать менять элемент.

Атрибут (событие) **onBlur** позволяет вызвать какую-либо функцию JavaScript при потере элементом фокуса.

Атрибут (событие) **onSelect** позволяет вызвать какую-либо функцию JavaScript, когда пользователь выделяет текст в поле ввода текста или пароля.

Атрибут (событие) **onChange** позволяет вызвать какую-либо функцию JavaScript, когда элемент теряет фокус и его значение изменилось с момента предыдущего получения фокуса.

Тег **<textarea>** позволяет визуализировать многострочное поле ввода текста.

Вид: <textarea>…</textarea> (блочный элемент)
Индивидуальные атрибуты: name, rows, cols, readonly, onFocus, onBlur, onSelect, onChange
Стандартное отображение: поле для ввода текста

Обязательные атрибуты **rows** и **cols** задают, соответственно, количество строк и столбцов (символов) отображаемой обозревателем Интернет области ввода. Эти атрибуты являются скорее рекомендацией обозревателю, чем строгим ограничением; пользователь может ввести больше текста, чем позволяют указанные атрибуты, поэтому обозреватели обеспечивают прокрутку поля ввода по горизонтали и вертикали. Теоретически обозреватели могут принимать ввод текста любого размера, но на практике этот размер обычно ограничен объемом в 32 или 64 кбайт. Необязательный атрибут **readonly** запрещает изменение первоначального содержимого поля.

Атрибуты (события) **onFocus**, **onBlur**, **onSelect**, **onChange** обрабатываются так же, как и в теге **<input>**.

Тег **<select>** создает управляющий элемент формы, предназначенный для выбора пользователем опций из списка. Его содержимым могут быть только теги **<option>**, которые определяют соответствующий список опций. Он может содержаться в любых блочных и текстовых элементах.

Вид: <select>…</select> (текстовый элемент)
Индивидуальные атрибуты: name, size, multiple, onFocus, onBlur, onChange
Стандартное отображение: в зависимости от значения атрибута size

Атрибут **name** задает имя данного управляющего элемента. Атрибут **size** указывает обозревателям размер видимой части списка (по умолчанию – выпадающий список). Если общее количество опций больше, то обозреватель обеспечивает прокрутку окна списка, чтобы обеспечить доступ ко всем опциям.

По умолчанию пользователь может выбрать только одну опцию из списка. Чтобы разрешить выбор нескольких опций одновременно, следует указать атрибут **multiple** (только имя атрибута без ввода какого бы то ни было значения). В этом случае каждая из выбранных опций передается при пересылке формы как отдельная пара имя/значение.

Тег **<option>** позволяет задать каждый элемент списка (по аналогии с тегом **<li>**):

Вид: <option>…</option>
Индивидуальные атрибуты: name, value, selected
Стандартное отображение: в зависимости от значения атрибута size

Атрибут **value** задает имя элемента, которое используется при пересылке формы. Если этого атрибута нет, то именем опции считается содержимое данного элемента. Атрибут **selected** указывает, что данная опция является выбранной в момент визуализации меню. Меню может содержать только одну опцию с этим атрибутом, если у него нет атрибута **multiple**.

Рассмотрим пример построения формы:

```
<html><body><form action="1.php" method="post" name="osebe">
<table border="0">
        <tr><td>Фамилия</td><td><input type="text" size="20"
name="surname" /></td></tr>
        <tr><td>Краткая биография</td><td>
            <textarea name="biograph">Указать:
            1.Предыдущее место работы
            2.Высшее образование</textarea>
        </td></tr>
```

```
        <tr><td>Выберите область деятельности</td><td>
            <select name="obldeyat">
              <option value="ed">образование</option>
              <option value="fin" selected>финансы</option>
            </select>
        </td></tr>
</table>
<input type="submit" value="Отправить">
</form></body></html>
```

В результате визуализации будет отображена форма, представленная на рис. 2.3.

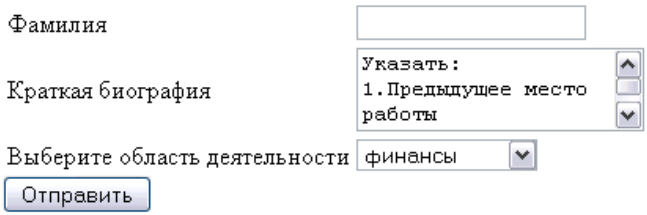


Рис. 2.3. Результат визуализации формы

2.10. Общие атрибуты

После того как стало очевидно, что за HTML должно остаться только логическое форматирование (функции экранного форматирования перенесены в CSS) и возникла потребность в организации взаимодействия с пользователем на странице (обозреватели начали обрабатывать конструкции JavaScript), в стандарт HTML были добавлены следующие общие для каждого тега атрибуты:

- **id** (identification) – персонализирующий атрибут в пределах документа, используется для назначения стилей, в качестве закладки (см. п. 2.3.), позволяет управлять атрибутами тега и его содержимым через JavaScript, в том числе извлекать данные. Исходя из перечисленного выше, должен назначаться только одной конкретной копии тега. Если одинаковый **id** будет присвоен разным

тегам, то это в большинстве случаев вызовет сбой как в оформлении страницы, так и при работе сценариев клиента. Атрибут игнорируется для тегов, употребляющихся в заголовке HTML-документа;

- **class** – атрибут определяет, что элемент принадлежит к определенному классу (классам), что позволяет визуализировать элемент в соответствии с заранее заданными правилами. Атрибут игнорируется для невизуализируемых тегов;
- **style** – атрибут позволяет управлять визуализацией данной конкретной копией тега, задав в качестве значений атрибута инструкции CSS;
- **title** – атрибут, описывающий краткое содержание элемента. Он часто выводится обозревателями как всплывающая подсказка (tooltip), которая выводится на экран, когда курсор помещается на данный элемент. Атрибут особенно полезен в элементах **<a>**, **<img>** и **<object>**, в которых он указывает на титул связанного или вложенного ресурса;
- атрибуты, которые определяют реакцию данного элемента на различные события (табл. 2.3).

Таблица 2.3

Названия атрибутов и реакция на события

Имя события	**Происходит**
onClick	при щелчке кнопки мыши на элементе
onDblClick	при двойном щелчке кнопки мыши на элементе
onMouseDown	при нажатии кнопки мыши на элементе
onMouseUp	при отпускании кнопки мыши на элементе
onMouseOver	при попадании курсора мыши на элемент
onMouseMove	при движении курсора мыши по элементу
onMouseOut	при попадании курсора мыши за пределы элемента
onKeyPress	при нажатии и отпускании клавиши на элементе
onKeyDown	при нажатии клавиши на элементе
onKeyUp	при отпускании клавиши на элементе

Задачи для самоконтроля

1. Создать страницу, на которой были бы представлены все категории HTML-элементов.
2. Создать кольцо из трех HTML-документов. Из каждой страницы можно попасть только на одну из подготовленных страниц. Ссылки должны идти с изображения, из текста внутри таблицы и с заголовка.
3. Сделать страницу с автобиографией, на которой разместить свою фотографию и сделать как минимум три раздела: родители, школьный и вузовский периоды жизни.

Раздел 3. Каскадные таблицы стилей (CSS)

3.1. Основные определения

CSS (Cascading Style Sheets – каскадные таблицы стилей) – одна из базовых технологий в современном Интернете. Сегодня уже редко можно встретить сайт, свёрстанный без применения CSS.

CSS-код – это список инструкций для обозревателя Интернет, описывающих, как и где визуализировать элементы HTML-документа.

В печатном деле вопрос оформления книги решается на ранних этапах её производства. Очевидно, что изменить базовые параметры оформления уже напечатанной книги не представляется возможным

В случае с сайтами это не так. Содержимое страницы благодаря логическому форматированию текста почти не связано с его экранным отображением. Обновив всего одну строку в css-стилях, дизайнер может радикально изменить оформление многих тысяч страниц сайта, придав шрифту всех заголовков, скажем, зелёный цвет, переместив блок новостей в другой угол или сменив фон страниц.

В декабре 1996 г. W3C стандартизовал первый уровень каскадных таблиц стилей (CSS1), который определил правила описания стилей визуального отображения элементов HTML-документов. В мае 1998-го консорциумом был принят стандарт второго уровня таблиц стилей (CSS2), который существенно расширил возможности таблиц стилей. Основные особенности CSS2:

- стили возможно применить к любым структурированным документам. На сегодня таковыми являются HTML-документы и XML-приложения;
- стало возможно дифференцировать стили для разных устройств: для принтеров, синтезаторов речи и др.

Вид CSS-инструкции

Все CSS-инструкции состоят из селектора и блока объявлений (заключённого в фигурные скобки). Внутри блока объявлений (в фигурных скобках) может находиться одно или несколько объявлений, разделённых точкой с запятой. Объявление – это строка,

составленная из css-свойства, и его значения, разделенные двоеточием. CSS не различает регистр символов.

Рис. 3.1. Схема CSS-инструкции

Пример CSS-инструкции:

```
p {
        text-align: justify; /* форматирование по ширине */
        text-indent: 10px;
}
```

Конструкция **/* что-то */** позволяет вставлять комментарии в CSS-инструкции, которые никак не обрабатываются обозревателем.

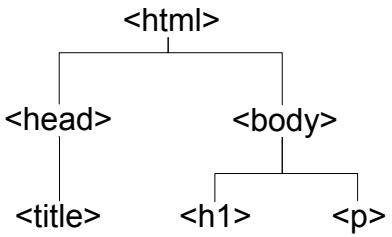


Рис. 3.2. Дерево тегов

Также необходимо учитывать, как происходит определение инструкций по умолчанию (когда инструкции не заданы). Обозреватель выстраивает дерево тегов (рис. 3.2.) в соответствии с учетом

вложений тегов друг в друга и на основании структуры данного дерева производит наследование инструкций определенных уровнем выше (у предков). Если на уровне выше нет явного объявления свойства, то обозреватель поднимается до самого верха, пока не найдет предка, у которого свойство объявлено, либо возьмет настройки, заданные пользователем в настройках обозревателя, если свойство нигде не определено.

Селекторы

Селектор – это часть CSS-инструкции, которая указывает, к каким элементам его применять. Рассмотрим три основных вида селекторов (элемента, класса и идентификатора) и способы их совмещения.

Селектор элемента совпадает с названием тега данного элемента:

```
div {color:red;} /* применяется ко всем элементам div в документе */
```

Селектор класса позволяет настроить визуализацию для всех тегов, которые подключили себе данный класс через атрибут **class**. Описание селектора класса начинается с точки после которой идет имя класса:

```
.alignleft {text-align: left;} /* рекомендуется называть класс так, чтобы было понятно, как изменится визуализация тега */
```

Селектор идентификатора позволяет настроить визуализацию для одного конкретного тега, для которого определен идентификатор через атрибут **id**. Описание селектора класса начинается с диеза (#), после которого идет имя идентификатора:

```
#leftmenu {margin:0;}
```

С помощью составных селекторов можно ограничивать область действия селекторов или присваивать один и тот же параметр визуализации сразу нескольким селекторам:

```
p.alignleft {text-align: left;} /* определяет параметры визуализации только элементов p класса alignleft */
```

```
h1#leftmenu {margin:0;} /* определяет параметры визуализации только в том случае, если идентификатор leftmenu присвоен тегу h1*/

td em {color:green} /* применяется ко всем элементам td и em в документе */
```

Приоритеты способов визуализации

На рис. 3.3 изображена схема определения важности параметров визуализации. То есть, если для тега задан атрибут style, указанные в нем свойствам будут иметь максимальный приоритет при визуализации. Если данный атрибут не определен, то приоритет переходит к свойствам, описанным для идентификатора и т.д.

Рис. 3.3. Приоритет подчиненности свойств при визуализации элемента

Оптимизация объявлений

В CSS встроены различные механизмы оптимизации объявлений, позволяющих сделать их либо более читаемыми, либо более компактными.

Например, есть общее свойство **margin**, которое позволяет задать крайнюю границу блока. Если мы пишем

```
p {margin:0}
```

то говорим обозревателю, что граница совпадает с рамкой блока, причем со всех четырех сторон. Данная запись эквивалентна следующим:

```
p {margin:0 0}
p {margin:0 0 0 0}
```

Перед нами типичный пример задания более компактного оформления. Причем пример с двумя нулями также является примером компактного написания. Но в данном случае первая цифра обозначает отступы сверху и снизу, а вторая – слева и справа.

Однако существуют и составные свойства:

```
p {margin-top:50px}
```

Составные свойства позволяют сделать инструкции более читаемыми. Вышеуказанный код эквивалентен записи:

```
p { margin:50px 0 0 0}
```

В этом случае надо вспоминать, что обозначает первое значение свойства, когда понадобится что-либо изменить. Хотя общее правило достаточно простое. Первое значение отступ сверху, а далее по часовой стрелке, т.е. отступ справа, отступ снизу и, наконец, отступ слева.

Составные свойства содержат в себе (не всегда)наименование общего свойства, а далее, через дефис, перечисляются детализирующие понимание части свойства.

Размеры

Размеры указывают на вертикальную или горизонтальную величину чего-либо. Размер задается как число, за которым следует единица измерения. Если размер равен 0, то единицу измерения можно не указывать.

Единицы измерения подразделяются на абсолютные и относительные. Абсолютные единицы измерения задают точный физический размер, а относительные – указывают размер относительно другого размера. Их описания сведены в табл. 3.1 и 3.2.

Таблица 3.1

Абсолютные единицы измерения

In	Дюймы (1 дюйм = 2,54 см = 25,4 мм = 72 точки = 6 пик)
cm	Сантиметры (1 см = 10 мм = 0,39 дюйма = 2,36 пики = 28,35 точки)

Окончание табл. 3.1

mm	Миллиметры (1 мм = 0,1 см = 0,039 дюйма = 0,24 пики = 2,84 точки)
pt	Точки (1 точка = 1/12 пики = 1/72 дюйма = 0,035 см = 0,35 мм)
pc	Пики (1 пика = 12 точек = 1/6 дюйма = 0,423 см = 4,23 мм)

Таблица 3.2

Относительные единицы измерения

em	Размер (font-size) соответствующего шрифта
ex	Высота строчных букв (x-height) соответствующего шрифта
px	Пиксели (размер зависит от устройства отображения)
%	Процент от размера

Абсолютные единицы измерения применимы только тогда, когда нам известны точные физические размеры устройства отображения (например, экрана дисплея или страницы принтера). Поэтому в большинстве случаев используются относительные единицы, назначение которых стоит пояснить подробнее.

Единицы em и ex основываются на размере шрифта того элемента, к которому относится данная декларация. При этом em задает размер шрифта, т. е. размер его наибольшей буквы (обычно это буква 'M', отсюда аббревиатура em), а ex – высоту строчных букв шрифта (обычно это высота буквы 'x', отсюда английское название x-height и аббревиатура ex). С другой стороны, единица px основана на размере пикселя устройства отображения (обычно это дисплей). Пиксель – точка дисплея и ее размер зависит как от физических размеров экрана, так и его разрешения: пиксель на экране с разрешением 640x480 будет больше, чем на экране с разрешением 1280x1024. Примеры задания размеров:

```
h1 {margin: 0.5em }
h1 {text-indent: 1ex }
h3 {font-size: 12px }
h1 {margin: 0.5in }
h2 {line-height: 3cm }
h3 {word-spacing: 4mm }
```

```
h4 {font-size: 12pt }
h4 {font-size: 1pc }
```

3.2. Включение CSS в HTML-документ

Для включения CSS в HTML-документ можно воспользоваться одним из четырех способов либо их комбинацией:

- применить внешние стили (в виде отдельного текстового .css-файла) с помощью тега **<link>**;
- встроить стили непосредственно в HTML-документ (в виде блока css-текста) с помощью тега **<style>**;
- применить внешние стили с помощью директивы **@import**;
- применить inline-стиль, т.е. назначить стиль конкретному HTML-элементу непосредственно в документе, с помощью HTML-атрибута **style**.

При подключении внешних таблиц стилей рекомендуется файлам давать расширение css, чтобы упростить задачу и серверному, и клиентскому программному обеспечению в идентификации содержания файла.

Внешние стили (external style sheets)

Применяются с помощью тега **<link>**, который должен располагаться исключительно внутри тега **<head>**:

```
<link rel="stylesheet" type="text/css" href="styles.css" media="all">
```

Встретив в HTML-документе этот тег, браузер загрузит с сайта CSS-файл (в нашем случае – styles.css) и применит к документу содержащиеся в нём стили. Файл должен содержать только CSS-инструкций.

Внешний файл со стилями удобен тем, что одни и те же стили можно применять ко множеству документов на сайте, достаточно в каждом из них всего лишь вписать одну строку с элементом **<link>**.

Рассмотрим более подробно тег **<link>**:

Вид: <link>
Индивидуальные атрибуты: rel, type, href, media
Стандартное отображение: нет

Тег **<link>** определяет взаимосвязь между документами.

Каждый тег **<link>** должен содержать атрибут **rel** и **href**.

Атрибут **rel** определяет тип ссылки, а атрибут **href** адрес ссылки. Указанный ниже код означает, что документ glossary.html является глоссарием терминов для текущего документа.

```
<link rel="glossary" href="glossary.html">
```

Атрибут **type** сообщает обозревателю, как обрабатывать подключаемый файл. Для CSS файлов его значение должно указываться как text/css.

Атрибут **media** указывает того, к отображению на каких устройствах применяется данная таблица стилей:

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

В табл. 3.3 приведены значения, которые может принимать параметр **media**. Допускается написание нескольких значений через запятую.

Таблица 3.3

Имена устройств отображения

all	Все типы устройств
aural	Синтезатор речи
braille	Тактильное устройство Брайля для слепых
embossed	Печатающее устройство Брайля
handheld	Переносное устройство (например, пейджер)
print	Принтер
projection	Проектор
screen	Графический дисплей
tty	Неграфический дисплей (терминал, телетайп)
tv	Телевизор

Таблицы стилей документа (document style sheets)

Таблицы стилей документа называются так потому, что располагаются непосредственно в HTML-документе и применяются только к нему. Иногда называются embedded style sheet (встроенная таблица стилей). В этом случае CSS-инструкции располагаются между открывающим и закрывающим тегами **<style>**:

Вид: <style> … </style>
Индивидуальные атрибуты: type
Стандартное отображение: нет

Атрибут **type** обязателен для указания, он подтверждает что инструкции по визуализации оформлены именно с применением CSS. Для этого используется следующее значение атрибута: text/css. Пример использования тега:

```
<style type="text/css">
        h1 {text-align: center;}
</style>
```

Сам тег **<style>** (в отличие от **<link>**) может находиться в любой части документа, но обычно его размещают внутри тега **<head>**, чтобы обозреватель Интернет раньше начал корректно визуализировать HTML-документ на экране.

Подключение внешних стилей через директиву

Директива **@import** позволяет включать в свою таблицу стилей другие таблицы стилей. Она должна содержать адрес импортируемой таблицы стилей. Следующие две директивы эквивалентны и демонстрируют правила написания данной директивы:

```
@import "mystyle.css";
@import url(styles.css);
```

Директива **@import** может содержать список названий устройств отображения, к которым должна применяться данная таблица стилей, разделенных запятыми. Например:

```
@import url("fineprint.css") print;
@import url("bluish.css") projection, tv;
```

Если списка названий устройств нет, то предполагается, что он равен **all**, т.е. импортируемая таблица стилей применима ко всем устройствам.

Директивы **@import** должны располагаться в элементе **<style>** или в css-файле перед первым правилом и не могут находиться внутри текстового блока; в противном случае они игнорируются обозревателем.

Стили, подставляемые в строку (inline styles)

Иногда нужно назначить стиль отдельному элементу на странице, не применяя внешних стилей и тега **<style>**. Типичный случай – элемент встречается единожды в документе или на сайте, но требует особого оформления. В этом случае возможно воспользоваться атрибутом **style**:

```
<p style="color:yellow; background-color:black">Это абзац, в котором шрифт визуализируется желтым цветом на черном фоне, других таких на сайте нет</p>
```

Как видно из примера, внутри атрибута **style** можно написать несколько CSS-объявлений, разделённых точкой с запятой, фигурные скобки не используются.

3.3. Шрифт

Общее свойство **font** позволяет полностью определить стиль шрифта. Последовательность указания значений роли не играет. Можно указывать не все значения.

Рассмотрим составные свойства, значения используются и в общем свойстве.

Свойство **font-family** задает список имен семейств шрифтов, применяемых при отображении содержимого элемента. Этот список состоит из имен семейств шрифтов, разделенных запятыми. Имена семейств располагаются в порядке предпочтения. Например, свойство

```
font-family: Verdana, Arial, sans-serif;
```

следует читать так: «использовать шрифт Verdana; если его нет, то использовать шрифт Arial; если его нет, то использовать родовой шрифт sans-serif». Такой список необходим, поскольку заранее не известно, какие именно шрифты установлены на компьютерах у пользователей.

Имя семейства шрифтов может быть задано двумя способами:

1) *имя-семейства* задает название семейства шрифтов. Если это название содержит пробелы, то оно должно быть заключено в кавычки или апострофы;
2) *родовое-имя* задает одно из следующих имен: serif, sans-serif, cursive, fantasy и monospace.

Для достижения наибольшей совместимости рекомендуется задавать родовое имя шрифта последним в списке. В этом случае, если обозреватель не найдет на компьютере-клиенте ни одного из заданных шрифтов, он использует свой родовой шрифт с заданным именем.

Свойство **font-style** задает стиль шрифта, применяемого для отображения содержимого элемента. Оно может иметь значения *normal* (обычный шрифт) и *italic* (курсивный):

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

Свойство **font-weight** задает степень жирности шрифта, применяемого для отображения содержимого элемента. Оно может принимать следующие основные значения: *normal* (обычный шрифт) и *bold* (полужирный):

```
h1 {font-style:normal;}
```

Свойство **font-size** задает размер шрифта, применяемого для отображения содержимого элемента. Оно может задаваться несколькими способами:

Значения

```
xx-small | x-small | small | medium | large | x-large | xx-large
```

задают абсолютный размер шрифта (от наименьшего к наибольшему).

Значения

```
smaller | larger
```

задают размер шрифта относительно родительского шрифта (меньше или больше). Также может быть задан абсолютный или относительный (относительно шрифта по умолчанию) размер шрифта.

3.4. Текст

Для оформления текста общее свойство не предусмотрено, однако существуют составные.

Свойство **text-indent** задает отступ первой строки (красную строку) при отображении блочных элементов. Отступ может быть отрицательным и задается абсолютным либо относительным (в процентах относительно ширины блока) способом:

```
h2 {text-indent:1.2em;}
```

Свойство **text-align** задает выравнивание текста для блочных элементов и может принимать следующие значения:

left	выравнивание по левому краю,
right	выравнивание по правому краю,
center	выравнивание по центру,
justify	выравнивание по ширине.

```
p {text-align:center;}
```

Свойство **text-decoration** позволяет оформить текст следующим образом:

none	обычный текст,
line-through	перечеркнутый текст,
overline	надчеркнутый текст,
underline	подчеркнутый текст.

```
a:link { text-decoration: none}
/* убирает подчеркивание с ссылок */
```

3.5. Цвет и фон

Правила визуализации цвета

Цветовые значения (далее обозначаются как «цвет») могут задаваться либо шестнадцатеричным числом со значением от 00 до FF с префиксом "#" вида "#rrggbb", определяющим RGB-код цвета (большинство воспринимаемых человеком цветом можно получить смешиванием основных трех цветов – красного, зеленого и синего, соответственно в формуле цвета rr = количество красного в цвете, gg = количество зеленого в цвете и bb = количество синего в цвете), либо одним из 16 символических имен, приведенных в табл. 3.4. CSS допускает использование краткой формы RGB-кодов вида "#rgb"; при этом краткая форма преобразуется в полную повторением цифр, а не добавлением нулей. Иными словами, код #fa1 соответствует полному коду #ffaa11. Кроме того, RGB-код цвета может быть задан в десятичной и процентной системах счисления конструкцией rgb(r,g,b), где r, g и b принимают значения в диапазоне от 0 до 255 или от 0% до 100%.

Таблица 3.4

Базовые цвета HTML

black (#000000) (0, 0, 0)	silver (#C0C0C0) (192, 192, 192)	gray (#808080) (128, 128, 128)	white (#FFFFFF) (255, 255, 255)
maroon (#800000) (128, 0, 0)	red (#FF0000) (255, 0, 0)	purple (#800080) (128, 0, 128)	fuchsia (#FF00FF) (255, 0, 255)
green (#008000) (0, 128, 0)	lime (#00FF00) (0, 255, 0)	olive (#808000) (128, 128, 0)	yellow (#FFFF00) (255, 255, 0)
navy (#000080) (0, 0, 128)	blue (#0000FF) (0, 0, 255)	teal (#008080) (0, 128, 128)	aqua (#00FFFF) (0, 255, 255)

Цвет текста

Цвет текста задает свойство **color**. Нижеследующие декларации задают один и тот же красный цвет для содержания элемента **em**:

```
em {color:red }                    /* название цвета */
em {color:#f00 }                        /* #rgb */
em {color:#ff0000 }                     /* #rrggbb */
em {color:rgb(255,0,0) }                /* целые в диапазоне 0 до 255
*/
em {color:rgb(100%,0%,0%) } /* действительные в диапазоне от
0.0% до 100.0% */
```

Управление фоном

Общее свойство **background** позволяет определить стиль фона как для документа в целом (рекомендуется использовать тег **<body>**), так и отдельных элементов. Последовательность указания значений роли не играет. Можно указывать не все значения. Фон элемента распространяется на всю занимаемую элементом область.

Рассмотрим составные свойства, значения используются и в общем свойстве.

Фон элемента может задаваться либо цветом (**background-color**), либо изображением (**background-image**, **background-repeat** и **background-position**).

Свойство **background-color** оформляется аналогично свойству **color**:

```
em {background-color:red }
```

Свойство **background-image** указывает обозревателю адрес изображения, которое необходимо использовать как фон элемента при отображении:

Свойство **background-repeat** указывает условия повтора изображения с помощью следующих значений:

repeat	изображение повторяется по горизонтали и вертикали,
repeat-x	изображение повторяется только по горизонтали,
repeat-y	изображение повторяется только по вертикали,
no-repeat	изображение не повторяется: отображается только одна его копия.

Свойство **background-position** задает позицию фонового изображения при отображении. Его можно определять как в абсолютных, так и относительных велечинах (процентах). Тем не менее

рассмотрим человекоориентированные возможности по определению позиции изображения:

top	прикрепить изображение к верхнему краю элемента,
bottom	прикрепить изображение к нижнему краю элемента,
left	прикрепить изображение к левому краю элемента,
right	прикрепить изображение к правому краю элемента,
center	отцентрировать изображение (по вертикали или горизонтали).

Значения употребляются по одиночке или парами, в следующих комбинациях:

top left = left top
top = top center = center top
right top = top right
left = left center = center left
center = center center
right = right center = center right
bottom left = left bottom
bottom = bottom center = center bottom
bottom right = right bottom

Общий пример инструкции:

```
body {
        background-image: url("river.png");
        background-position: bottom right;
        background-repeat: no-repeat;
}
```

3.6. Оформление блоков

CSS предусматривает следующую модель оформления блоков: размер любого блока складывается из ширины и высоты области содержимого, отступов, толщины рамок и величины границ (рис. 3.4).

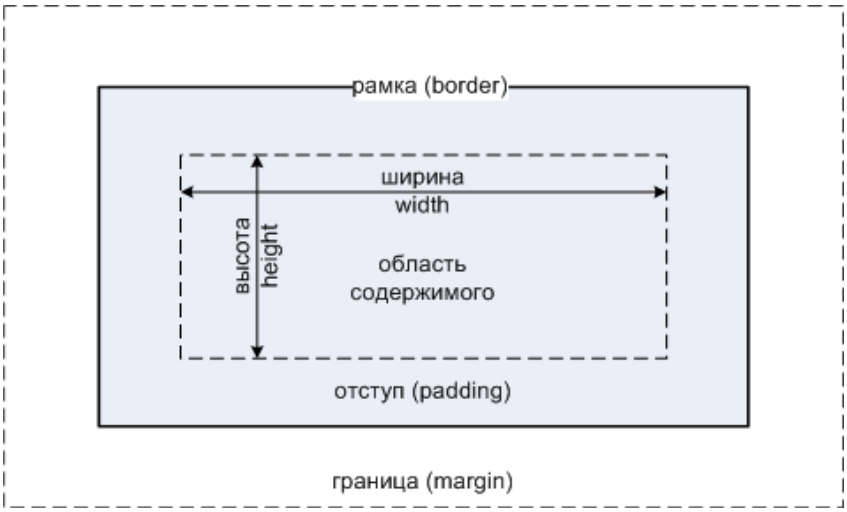


Рис. 3.4. Модель блока

В область содержимого (самый внутренний прямоугольник) располагается текст, мультимедийные объекты и другие блоки. Размер области содержимого можно регулировать через свойства **width** и **height**, которые могут принимать абсолютные или относительные (относительно размера блока-родителя) значения.

Отступ формирует расстояние между областью содержимого и рамкой. Эта область заливается фоном блока. Общее свойство отступа называется **padding**. Составные свойства: **padding-top**, **padding-bottom**, **padding-left** и **padding-right**. Из их названий ясно, какой из отступов они позволяют сделать. Правила подстановки значений аналогично описанному в п. 3.1 в подразделе «Оптимизация объявлений». Возможны как абсолютные, так и относительные (вычисляются от размера области содержимого) значения.

```
body {padding: 1em 2em 3em 4em }
/* top=1em, right=2em, bottom=3em, left=4em */
```

Рамка позволяет привнести декоративный элемент в оформление и, конечно, своей шириной влияет на размер блока.

Свойства рамки задают размер, цвет и стиль рамки объемлющего прямоугольника. Соответственно их можно разбить на следующие группы:

- размер рамки: **border-top-width**, **border-bottom-width**, **border-left-width**, **border-right-width** и **border-width**;
- цвет рамки: **border-top-color**, **border-bottom-color**, **border-left-color**, **border-right-color** и **border-color**;
- стиль рамки: **border-top-style**, **border-bottom-style**, **border-left-style**, **border-right-style** и **border-style**;
- общие: **border-top**, **border-bottom**, **border-left**, **border-right** и **border**.

Размер и цвет рамки определяются аналогично свойствам **padding** и **color**.

Стили рамки могут принимать следующие значения:

none нет рамки (влечет присваивание border-width значения 0),

dotted пунктирная рамка (выводится точками),

d*ashed* штриховая рамка (выводится короткими отрезками),

solid сплошная рамка (выводится сплошной линией стоит по умолчанию),

double двойная рамка (выводится двойной сплошной линией),

groove рамка изображается в виде трехмерной выемки,

ridge противоположность groove. Рамка изображается в виде трехмерного выступа,

inset рамка изображается в виде трехмерной врезки,

outset противоположность inset. Рамка изображается в виде трехмерного вырезки.

```
h1 { border-right-style: double }
```

Общее свойство границы называется **margin**. Граница позволяет отодвинуть рамку и область содержимого от других блоков и краев обозревателя. Область от границы до рамки всегда прозрачна (через нее виден фон, но свою заливку делать нельзя). Границу нельзя сделать видимой. Составные свойства: **margin-top**, **margin-bottom**, **margin-left** и **margin-right**. Из их название ясно, какой из отступов они позволяют сделать. Правила подстановки значений описаны в п. 3.1 подраздела «Оптимизация объявлений». Возможны как абсолютные, так и относительные значения.

3.7. Позиционирование элементов

CSS позволяет не только управлять видом блока, но также управлять его размещением на странице.

Свойство **display** позволяет либо управлять типом элемента, например из блочного делать текстовый или наоборот, либо скрыть его. Управление производится посредством присвоению свойству следующих значений:

block блочный элемент,
inline текстовый элемент,
none элемент и все его потомки игнорируются при отображении.

Свойство **position** позволяет определить одну из чеырех схем позиционирования блоков.

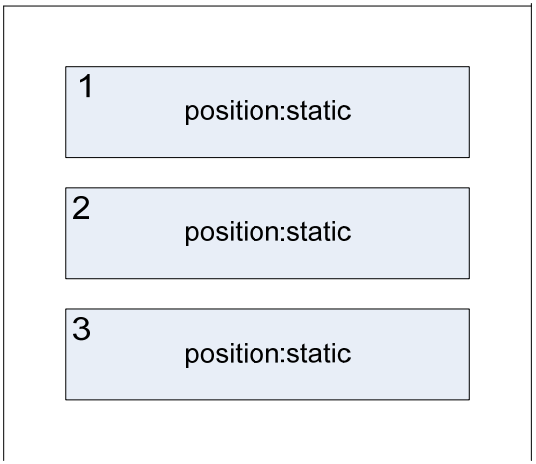


Рис. 3.5. Прямой поток

Способ по умолчанию – **static**, подразумевающий отсутствие какого бы то ни было специального позиционирования. То есть при данном способе блоки кладутся на странице или в другом блоке один за другим сверху вниз (рис. 3.5.). Такой порядок позиционирования называется «прямым потоком».

Значение **absolute** позволяет задать для блока абсолютное позиционирование в соответствии с заданным координатам относи-

тельно блока-родителя. При этом блок удаляется из того места, где он должен был бы быть в прямом потоке, а на его место поднимается следующий за ним блок (рис. 3.6). Такая процедура называется «исключением из потока».

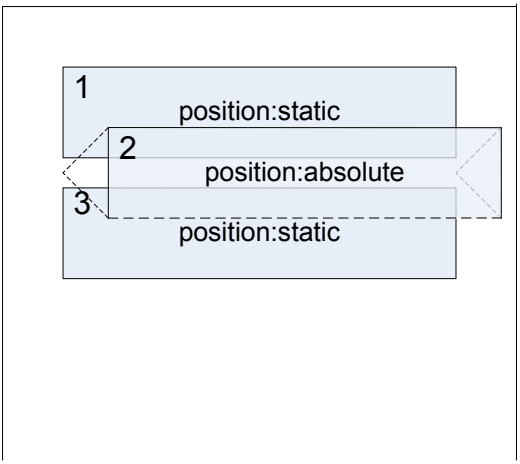


Рис. 3.6. Исключение блока из потока

Значение **fixed** аналогично значению **absolute**, но при этом при перемотке страницы блок всегда остается на заданных ему изначально координатах.

Значение **relative** позволяет вынуть блок из потока и разместить его в любом месте, но следующий за ним блок при этом не занимает освободившееся место (рис. 3.7).

Координаты вынутых из потока блоков задаются следующими свойствами: **top**, **right**, **bottom** и **left**. Их значения могут принимать как абсолютные, так и относительные (вычисляются от значений высоты или ширины блока) величины.

При этом необходимо достаточно четко представлять, относительно чего отсчитываются координаты. Страница формируется как своеобразный перевернутый «стакан», начинающийся от верха окна, ограниченный с боков и бесконечно продолжающийся вниз. Если все блоки статические, то они один за другим вставляются обозревателем Интернет в этот стакан (см. рис. 3.4). Если в этом

потоке появляется позиционированный блок, то его координаты вычисляются от сторон этого самого «стакана».

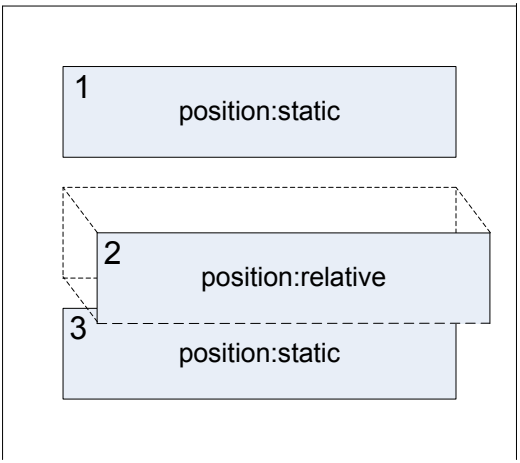


Рис. 3.7. Относительное позиционирование

При этом позиционированный блок сам внутри себя создает такой же «стакан», и все его «дети» (блоки, находящиеся у него внутри) позиционируются уже относительно него, а не относительно окна. И внутри него происходит то же самое: любой позиционированный блок создает внутри себя такой «стакан».

В терминах CSS этот «стакан» называется «содержащим блоком» (containing block).

Абсолютное позиционирование

Итак, чтобы расположить блок абсолютно, ему надо задать нужный тип позиционирования и координаты:

```
#textbody {
        position:absolute;
        top:0; right:10px; left:100px; bottom:100px;
}
```

Координаты означают расстояние блока от краев:

- «top:0» – бокс прижат к верхнему краю,
- «right:10px» – отстоит на 10 пикселов от правого края и т.д.

Любая из координат необязательна. В случае, если координаты не задают вертикального или горизонтального положения, то оно остается таким же, каким было бы без позиционирования. То есть в случае, когда у нас есть два произвольных бокса один за другим "box1" и "box2":

```
<div id="box1"> ... </div>
<div id="box2"> ... </div>
```

... и второй мы позиционируем так:

```
#box2 {position:absolute; left:150px;}
```

... то по вертикали он останется прямо под первым блоком, а по горизонтали будет отстоять от левого края на 150 пикселов.

Относительное позиционирование

Чаще всего относительное позиционирование используется без задания смещений. В этом случае он ведет себя как обычный статический блок, но поскольку он таким не является, то создает внутри себя содержащий блок, относительно которого будут позиционироваться блоки внутри него.

Рассмотрим пример. Пусть у нас есть три блока: «шапка», «основной текст» и «подвал», а внутри «основного текста» лежит блок «реклама»:

```
<html><body>
<div id="header"> шапка </div>
<div id="contents">
        <div id="adv"> реклама </div>
        <p>Основной текст</p>
```

```
</div>
<div id="footer">подвал</div>
</html></body>
```

При этом высота заголовка не зафиксирована, а значит, точно не известна. Блоки эти статические, идут один за другим, и какая бы высота у заголовка ни была, основной текст начинается прямо под ним.

А теперь нам хочется блок с рекламой расположить так, чтобы он был точно в правом верхнем углу основного текста (рис. 3.8).

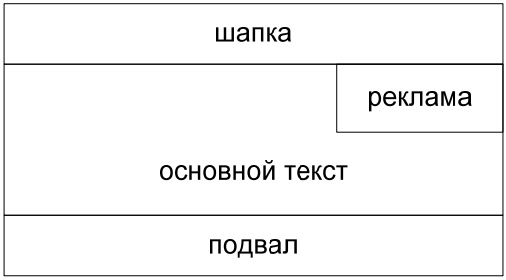


Рис. 3.8. Пример размещения блоков

Напишем ему:

```
#adv {position:absolute; top:0; right:0;}
```

Однако это не решение, поскольку содержащим блоком для него сейчас является все окно, и блок с рекламой уедет поверх заголовка в правый верхний угол обозревателя. Отодвинуть его ниже заголовка не получится, так как высота последнего не зафиксирована и при разных разрешениях экрана может гулять.

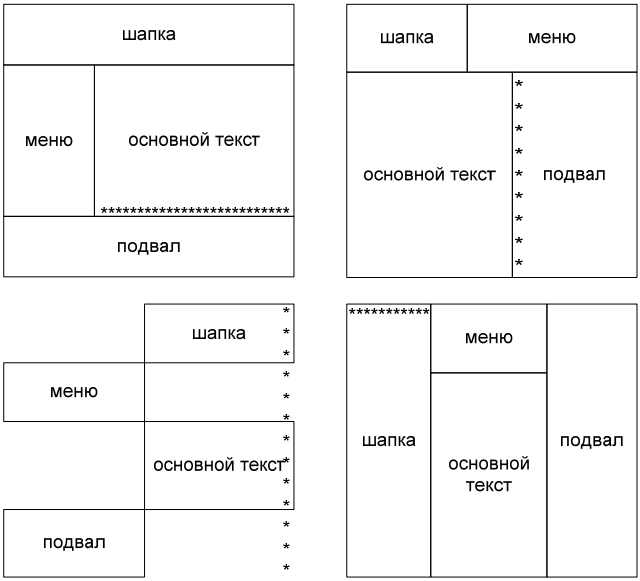
Можно попробовать сделать блок основного текста тоже абсолютным, тогда он станет содержащим блоком. При этом он сам выдернется из потока и сверху на него наложится подвал страницы.

Это и есть один из случаев, когда требуется относительное позиционирование. То еесть основной текст никуда не девается из потока, но при этом становится содержащим блоком:

```
# contents {position:relative;}
```

Задачи для самоконтроля

1. Подготовить один HTML-документ и к нему четыре таблицы со стилями, позволяющими сделать следующую визуализацию (вместо * вставить любое фоновое изображение):



2. Возьмите по адресу http://vesna.yandex.ru/ несколько произвольных текстов и с помощью CSS оформите макет страницы, в котором есть:
   - шапка (заголовок первого уровня, под который подложен фон по длине на всю страницу);
   - оглавление (ссылки на разделы данного документа);
   - разделы (заголовки второго уровня) с текстами (отформатировано по ширине, красная строка). Каждый раздел выделен рамкой и отступами от

других разделов и краев страницы. Каждый заголовок раздела выделен фоном.

3. Сделайте для всех изображений HTML-документа пунктирную рамку по периметру, не зависящую от размера изображения.
4. Найдите и реализуйте способ преобразования списка (<ol> или <ul>) в строку, где элементы списка отделены друг от друга не новыми строками, а промежутками (пробелами).

РАЗДЕЛ 4. СЦЕНАРИИ КЛИЕНТА: ЯЗЫК JAVASCRIPT, МОДЕЛИ DHTML И DOM

4.1. Основные определения

JavaScript – объектно-ориентированный язык программирования, предназначенный для написания сценариев, работающих как на стороне клиента, так и на стороне сервера. Таким образом он не является «полноценным» языком программирования, а ориентирован на использование возможностей только той среды, в которой сценарии исполняются.

Обозреватель Интернет, работающий на компьютере-клиенте, обеспечивает среду, в которой JavaScript имеет доступ к объектам, представляющим собой окна, меню, диалоги, текстовые области, куки и ввод-вывод в HTML-документ. Кроме того, обозреватель Интернет позволяет присоединить сценарии на языке JavaScript к таким событиям, как загрузка и выгрузка страниц и графических образов, нажатие клавиш и движение мыши, выбор текста и пересылка форм (см. соответствующие общие атрибуты HTML-тегов в п. 2.10). При этом программный код сценариев только реагирует на события и поэтому не нуждается в главной программе. Набор объектов, предоставляемых обозревателем, известен под названием Document Object Model (DOM).

Язык программирования JavaScript был разработан Бренданом Эйком (Brendan Eich) в Netscape Communications как язык сценариев для обозревателей Netscape Navigator, начиная с версии 2.0. В дальнейшем к развитию этого языка подключилась корпорация Microsoft, чьи обозреватели Internet Explorer поддерживают JavaScript, начиная с версии 3.0. Версия Microsoft получила название JScipt, поскольку JavaScript является зарегистрированной маркой фирмы Netscape. В 1996 г. в ECMA решили стандартизировать этот язык, и в июне 1997-го была принята первая версия стандарта под названием ECMAScript (ECMA-262). В апреле 1998 г. этот стандарт был принят ISO в качестве международного под номером ISO/IEC 16262.

Структура программ

Программа (сценарий) на языке JavaScript – это текст, состоящий из операторов, блоков, т.е. взаимосвязанных наборов операторов, и комментариев. Операторы могут содержать переменные, константы и выражения. Блок объединяет набор операторов, заключенный в фигурные скобки {}. А комментарии делятся на два типа: однострочные (предваряются двумя дробями «//») и многострочные (выделяются открывающим элементом «/*» и закрывающим элементом «*/»).

```
function convertToMeters(inches) {
        // Следующие два оператора заключены в блок.
        meters = inches / 39.37;
        return meters;
}
// Следующие два оператора не образуют блока.
inches = 100;
km = convertToMeters(inches)/1000;
```

Как видно из примера, каждый оператор JavaScript начинается с новой строки (что не обязательно, но рекомендуется) и заканчивается точкой с запятой (что также не обязательно, если оператор является последним в строке).

Необходимо помнить, что в JavaScript регистр букв имеет значение, т.е. переменная **meters** не равна переменной **Meters**.

4.2. Переменные и массивы

В JavaScript различают четырк основных типа переменных:

1) числа (целые и дробные);
2) строки (при объявлении символы заключаются в парные или одинарные кавычки);
3) логические, т.е. имеющие два значения:
    - true или 1 или непустая строка,

- false или 0 или пустая строка;

4) объекты.

Массивы (Array) – самый распространенный объект, так как это и список всех гипертекстовых ссылок, и список всех изображений на странице, и т.д.

Массивы можно создать несколькими разными способами.

1. Через инициализатор массива:

```
colors = ["red","white","blue"]
```

2. Через конструктор массива:

```
new_array = new Array() // пустой массив
new_array5 = new Array(5) // пустой массив из 5 элементов
colors = new Array ("red","white","blue") // массив из 3 элементов: red, white, blue
```

Чтобы получить данные из i-го элемента массива, необходимо в качестве переменной использовать следующую конструкцию **colors[i]**. Первый элемент массива имеет номер **0**.

Чтобы присвоить данные используется та же конструкция:

```
colors[99] = black;
```

В примере не только присваивается значение 100-му элементу, но и проводится расширение размера самого массива до 100 элементов.

Число элементов массива определяется через метод **length**

```
alert(color.length);
```

4.3. Операции

В JavaScript используются стандартные C-подобные операции:

- бинарные (сложение «+», вычитание «−», умножение «*», деление «/»):

```
x = a+b // в переменную x попадает результат сложения переменных a и b
```

- унарные:
    - применяемы после переменных (инкремент «++» и декремент «--»)

```
a++; // то же самое, что и a=a+1
```

    - применяемые перед переменными (не «!», смена знака «-»)

```
-a; // то же самое, что и a=0-a
```

- присваивания (=, +=, -=, *=, /=):

```
a += b // то же самое что и a=a+b
```

- сравнения (меньше «<», больше «>», меньше либо равно «<=», больше либо равно «>=», равно «==», не равно «!=»):

```
a>b // возвращает true или false, т.е. больше а чем b или нет
```

- логические (и – «&&», или – «||»):

```
a || b // если хотя бы одна из переменных 1, то результат операции – 1
```

При использование операций тип результирующего значения (число или строка) определяется автоматически.

4.4. Операторы

Условный оператор

Условный оператор **if…else** позволяет проверить определенное условие и, в зависимости от его истинности, выполнить ту или иную последовательность операторов. Он имеет две формы:

```
if (условие) оператор1
if (условие) оператор1 else оператор2
```

Здесь условие – это любое выражение, значение которого может быть преобразовано к логическому типу, **оператор1** и **оператор2** –

любые группы операторов JavaScript; если эти группы содержат более одного оператора, то они должны быть заключены в фигурные скобки {}.

Первая форма оператора означает: если значение условия истинно, то выполняется **оператор1**; если оно ложно, то управление передается оператору, следующему за **if**.

Вторая форма оператора означает: если значение условия истинно, то выполняется **оператор1**, если оно ложно, то выполняется **оператор2**.

Пример использования условного оператора:

```
if (a == "молоко") {
        document.writeln("А у Вас ");
        document.writeln(a);
        document.writeln(" убежало ");
} else
        document.writeln("Не "+a+", а молоко");
```

Операторы цикла

Цикл – последовательность операторов, выполнение которой повторяется до тех пор, пока определенное условие не станет ложным. Далее рассмотрим два основных оператора цикла: **for** и **while**.

Оператор for

Оператор цикла **for** имеет вид

```
for (инициализация; условие; изменение) оператор
```

Здесь **инициализация** и **изменение** – любое выражения, **условие** – любое выражение, значение которого может быть преобразовано к логическому типу, **оператор** – любая группа операторов JavaScript; если данная группа содержит более одного оператора, то она должны быть заключена в фигурные скобки {}. Инициализация может содержать декларацию переменной.

Оператор **for** выполняется следующим образом:

1. Выполняется выражение инициализация (обычно это выражение инициализирует счетчик (счетчики) цикла).
2. Вычисляется значение выражения условие. Если оно ложно, то управление передается оператору, следующему за данным оператором.
3. Выполняется оператор.
4. Выполняется выражение изменение (обычно это выражение увеличивает или уменьшает счетчик (счетчики) цикла) и действия повторяются.

Данный оператор обычно используется в тех случаях, когда количество повторений цикла известно заранее. Например, следующий цикл присваивает каждому элементу массива его номер:

```
for (i = 0; i < a.length; i++) a[i] = i;
```

Оператор for…in

Оператор for…in выполняет заданные действия для каждого свойства объекта или для каждого элемента массива. Он имеет вид

```
for (переменная in выражение) оператор
```

Здесь **переменная** – декларация переменной, **выражение** – любое выражение, значением которого является объект или массив, **оператор** – любая группа операторов JavaScript; если эта группа содержит более одного оператора, то она должны быть заключена в фигурные скобки {}.

Оператор **for…in** выполняется следующим образом:

1. Переменной присваивается имя очередного свойства объекта или очередного элемента массива (это зависит от того, является значением выражения объект или массив).
2. Выполняется оператор.
3. Управление передается этапу 1.

При итерации массива переменной последовательно присваиваются значение первого, второго, …, последнего элемента массива. Однако при итерации свойств объекта невозможно предсказать, в каком порядке они будут присваиваться переменной: этот оператор гарантирует только то, что все они будут просмотрены.

Следующий сценарий создает новый ассоциативный массив **as**, а затем последовательно выводит все его значения на экран обозревателя:

```
var as = {"а" : "апельсин", "б" : "банан", "в" : "виноград"};
for (var key in as)
        document.write(key + ": " + as[key] + "<BR>");
```

Оператор while

Оператор цикла while имеет вид

```
while (условие) оператор
```

Здесь **условие** – любое выражение, значение которого может быть преобразовано к логическому типу, **оператор** – любая группа операторов JavaScript; если эти группа содержит более одного оператора, то она должны быть заключена в фигурные скобки {}.

Оператор **while** выполняется следующим образом:

1) вычисляется значение выражения условие. Если оно ложно, то управление передается оператору, следующему за данным оператором;
2) выполняется оператор и управление передается этапу 1.

Пример использования:

```
i=50;
while (i<100) i++;
```

4.5. Функции

Декларация функции

Функции являются одним из основных механизмов языка JavaScript; они охватывают ту область, которая в других языках программирования реализуется подпрограммами, процедурами и функциями. Функция в JavaScript – набор операторов, выполняющих определенную задачу.

Для того чтобы пользоваться функцией, необходимо её описать. Декларация функции имеет вид

```
function имя(аргументы) {
        операторы
}
```

Здесь **имя** – идентификатор, задающий имя функции, **аргументы** – необязательный список идентификаторов, разделенных запятыми, который содержит имена формальных аргументов функции, а **операторы** – любой набор операторов, называемый телом функции и исполняющийся при её вызове.

Рассмотрим следующий пример:

```
function quad(num) {
        return num * num;
}
```

Данная функция называется quad и имеет один формальный аргумент num. При её вызове вместо формального аргумента подставляется его фактическое значение, функция выполняет возведение его в квадрат и возвращает полученное число оператором return.

Переменные, декларированные в теле функции, локальны, т.е. недоступны вне ее тела.

Вызов функции

Важно понимать, что появление декларации функции в тексте сценария не означает ее немедленного выполнения; тело функции будет выполняться только тогда, когда какой-либо оператор будет содержать вызов этой функции. Например, функция из предыдущего примера может быть вызвана так:

```
var x = quad(5);
```

В результате переменная x получит значение 25.

Оператор return

Функции JavaScript могут (но не обязаны) возвращать значение. Для указания этого значения используется оператор return:

```
return выражение
```

Оператор прерывает выполнение функции и возвращает значение **выражения**. Функция, содержащая оператор **return**, должна вызываться как часть выражения присваивания:

```
x = 2 * quad(a).
```

Если вставить оператор **return** без выражения, то выполнение функции прерывается и возвращается значение **undefined.** Более того, такая функция должна вызываться как оператор. Например:

```
setColor(myColor);
```

Если тело функции не содержит оператора **return**, то ее выполнение завершается с выполнением последнего оператора тела и возвращается значение **undefined**.

4.6. Включение JavaScript в HTML-документ

Расположение внутри страницы

Для добавления JavaScript-кода на страницу, можно использовать тег **script**:

Вид: <script>…</script>
Индивидуальные атрибуты: type, src, charset
Стандартное отображение содержания: нет

Атрибут **type** указывает, на каком именно скриптовом языке написан программный код (кроме JavaScript, еще известны непопулярные и нестандартизованные JScript и VBScript). Рекомендуется указывать или «**application/javascript**», или «**text/javascript**».

Атрибут **src** позволяет задать путь к внешнему файлу со скриптами, а атрибут **charset** указать, какая кодировка для воспроизведения национального языка используется.

Пример скрипта, выводящего модальное окно с классической надписью «Hello, World!» внутри браузера:

```
<script type="text/javascript"> alert('Hello, World!'); </script>
```

Расположение внутри тега

Спецификация HTML описывает набор атрибутов, используемых для задания обработчиков событий. Пример использования:

```
<a href="delete.php"
onClick="return confirm('Вы уверены?');">Удалить</a>
```

Отделение от разметки

В приведённом примере при нажатии на ссылку функция confirm('Вы уверены?'); вызывает модальное окно с надписью «Вы уверены?», a return false; блокирует переход по ссылке. Разумеется, этот код будет работать только если в браузере есть и включена поддержка JavaScript, иначе переход по ссылке произойдёт без предупреждения.

Однако в рамках модели логического форматирования документов указанный выше пример необходимо использовать в крайнем случае, например при минимальном использовании JavaScript на странице, т.е. по аналогии с атрибутом **style** (см. п.3.2). Аналогом приведённого примера, при условии снабжения ссылки идентификатором **alertLink**:

```
<a href="delete.php" id="alertLink">Удалить</a>
```

может выступать, например, следующий фрагмент JavaScript:

```
window.onload = function() {
        var linkWithAlert = document.getElementById("alertLink");
        linkWithAlert.onclick = function() {
            return confirm('Вы уверены?');
        };
};
```

Вынесение в отдельный файл

Третья возможность подключения JavaScript – написать скрипт в отдельном файле, а потом подключить его к HTML-документу с помощью конструкции

```
<script type="text/javascript"
src="http://Путь_к_файлу_со_скриптом"></script>
```

Чтобы упростить задачу и серверному, и клиентскому программному обеспечению в идентификации содержания файла с JavaScript, рекомендуется давать таким файлам расширение js.

4.7. Объекты

Для создания механизма управления страницами на клиентской стороне используется объектная модель документа (DOM – Document Object Model). Суть модели в том, что каждому HTML-элементу соответствует объект, который характеризуется тройкой:

- свойства,
- методы,
- события.

DOM предоставляет удобный способ управления HTML-элементами из HTML-кода через инструкции JavaScript. Иными словами, обозреватель, загрузив страницу, строит дерево объектов (HTML-элементов) и видимым или невидимым для пользователя образом реагирует на изменения, производимые в нем средствами JavaScript.

Дерево объектов обозревателя Интернет

В DOM корнем дерева объектов является объект **window** (рис. 4.1), т.е. каждый объект приписан к тому или иному окну. Для обращения к любому объекту (его свойству) указывают полное или частичное имя этого объекта или его свойства, начиная с имени объекта, старшего в иерархии, в который входит данный объект.

Объект **navigator** хранит информацию об обозревателе Интернет, что позволяет обходить отдельные моменты несовместимости реализаций JavaScript, а также собирать статистику того, какие обозреватели применяют пользователи.

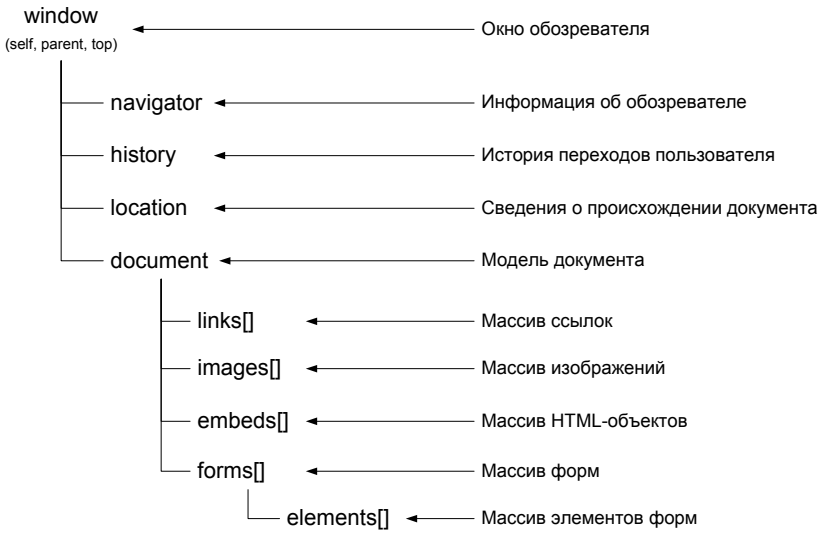


Рис. 4.1. Фрагмент дерева объектов

Объект **history** позволяет реализовывать переходы пользователей на уже посещенные ими страницы.

Объект **location** позволяет определить текущий адрес документа и организовать переход к любому другому ресурсу Интернет.

Объект window

В связи с тем, что объект **window** является корневым для дерева, принято допущение о том, что его имя можно опускать при обращении к объектам второго уровня, свойствам и методам объекта **window**. Например, вместо **window.location** допустимо писать просто **location**.

Рассмотрим основные методы объекта **window**.

Метод **open()** позволяет открывать новые окна, с заранее заданными характеристиками (размер, координаты, наличие панели инструментов, полосы скроллинга и т.п.).

Метод **open()** вызывается следующим образом:

```
window.open("URL", "имя_окна", ["атрибут_окна_1, …"]);
```

где строка **"URL"** – адрес ресурса, загружаемого в новое окно. Если URL не указан, окно открывается, но загрузки документа не происходит (кавычки и запятая должны присутствовать в любом случае).

С помощью аргумента **"имя_окна"** задают имя окна, которое не является его заголовком и должно быть написано с применением букв только латинского алфавита. В дальнейшем, используя данное имя, возможно изменять содержимое нового окна из текущего.

Аргумент **"атрибут_окна_1, …"** – список характеристик нового окна, задавать которые не обязательно: наличие меню, полос прокрутки, рамки, позволяющей изменять размер окна, данные о размерах окна и др. Характеристики перечисляются друг за другом через запятую.

Каждая характеристика окна представляется в виде атрибута с уникальным именем и может быть включена или выключена при помощи установок yes/no или 1/0. Подробнее – в табл. 4.1.

Таблица 4.1

Список атрибутов метода open()

Атрибут	Значение	Описание
copyhistory	[=yes|no] | [=1|0]	Сохранение истории загрузки документов в данное окно
Height	=pixelheight	Высота окна в пикселях
Location	[=yes|no] | [=1|0]	Наличие поля Location
Menubar	[=yes|no] | [=1|0]	Наличие меню
Resizable	[=yes|no] | [=1|0]	Разрешение на изменение размеров окна
Scrollbars	[=yes|no] | [=1|0]	Наличие линеек прокрутки
Status	[=yes|no] | [=1|0]	Наличие строки состояния
Toolbar	[=yes|no] | [=1|0]	Наличие панели инструментов
Width	=pixelwidth	Ширина окна в пикселях

По умолчанию атрибутам всегда присваивается значение yes, а размер нового окна (если он не задан) соответствует размеру предыдущего. Атрибуты указываются в произвольном порядке.

Метод **close()** позволяет закрыть окно, имя которого известно, или текущее.

Рассмотрим простейший пример, который позволяет открыть новое окно и закрыть его из окна-«родителя».

```
<html><head>
        <script type="text/javascript">
        function opWind(url) {
            newWin=window.open(url, "wind1", "width=200,
            height=100, resizable=no, scrollbars=no, menubar=no");
        }
        </script>
</head><body>
        <input type="button" value="Открыть окно"
        onClick="opWind('1.html')">
        <input type="button" value="Закрыть окно"
        onClick="newWin.close()">
</body></html>
```

Вновь открытое окно также может само себя закрыть, используя псеводним **self**. Рассмотрим листинг файла **1.html**:

```
<html><head><title> окно newWin</title></head>
<body style="text-align:center;">
        <span style="color:red; font-size:x-large;">
        это новое окно</span><br>
        <input type="button" value="закрыть окно"
        onClick="self.close()">
</body></html>
```

С помощью модальных окон, вызываемых методами **alert()**, **confirm()** и **prompt()** JavaScript позволяет уведомить пользователя о чем-либо и получить его реакцию на сообщение. Пока модальные окна не будут закрыты, пользователь не сможет продолжить работу с документом, загруженным в окно.

Метод **alert()** выводит окно с сообщением и кнопкой ОК, с помощью которой пользователь подтверждает, что он прочитал выведенное сообщение. Пример вызова:

```
alert('Добрый день');
```

Метод **confirm()** выводит окно с сообщением и кнопками [ОК] и [ОТМЕНА]. В зависимости от нажатой кнопки метод вернет 1 или 0 (true или false) соответственно. Пример использования:

```
if (confirm("Хочешь на сайт МИФИ?")) {
        location.href="http://mephi.ru";
} else {
        alert("А зря!");
}
```

Метод **prompt()** выводит окно с сообщением, полем ввода и кнопками [ОК] и [ОТМЕНА]. Возвращает введенное пользователем значение или ничего (null), если пользователь нажал [ОТМЕНА]. Пример использования:

```
alert('Вам '+ prompt('Сколько вам лет?', 20)+' лет!')
```

После запятой указывается значение по умолчанию, выводимое в поле ввода.

Последний блок методов позволяет выполнять программный код с задержкой или с определенной периодичностью.

Методы **setTimeout()** и **clearTimeout()** позволяют запустить и сбросить таймер:

```
<input type="button" onclick="timer_on()"
value="Запустить таймаут"/>
<input type="button" onclick="timer_off()"
value="Остановить отсчет"/>
<script>
        function callWin () { alert('Прошло 3 секунды') }
```

```
        function timer_on() {timeoutId = setTimeout(callWin, 3000)}
        function timer_off() {clearTimeout(timeoutId)}
</script>
```

Первый параметр функции **setTimeout()** – имя функции, которая будет вызываться, когда таймер остановится после заданного вторым параметром времени (в миллисекундах)

Методы **setInterval()** и **clearInterval()** вызывают какой-либо программный код через заранее заданный интервал времени и прерывать данный цикл:

```
<html><head><script type="text/javascript">
        var tmr;

        function si() {
            if (confirm('Прошло 5 секунд, остановить цикл?'))
                clearInterval(tmr);
        }
</script></head><body onLoad="tmr=setInterval(si, 5000);">
        Каждые 5 секунд будет появляться надпись
        с предложением прервать цикл.
</body></html>
```

Если переменную **tmr** не объявить как глобальную (до всех функций), то программа выполняться не будет. Имя функции указывается без скобок.

Объект Date

Объект Date позволяет объявлять переменные, содержащие календарную дату. В следующем примере создается переменная, в которую будет помещена дата создания HTML-документа.

```
LD = new Date(document.lastModified)
```

Методы LD.getDate, LD.getMonth, LD.getYear, LD.getHours, LD.getMinutes, LD.getSeconds позволяют извлечь из даты день, месяц, год, часы, минуты, секунды.

Следующий пример показывает, как организовать изображение работающих цифровых часов на странице:

```
<html><head><title>Часы в поле формы</title></head>
<body onLoad="myclock()">
<script type="text/javascript">
function myclock() {
        ndata=new Date() // Получить текущее время с компьютера клиента
        hours= ndata.getHours(); // Получить текущий час
        mins= ndata.getMinutes(); // Получить текущие минуты
        secs= ndata.getSeconds(); // Получить текущие секунды

        /* Привести изображение минут и секунд к удобному
        для восприятия человеком виду*/
        if (mins < 10) {mins = "0" + mins }
        if (secs < 10) {secs = "0" + secs }

        /* Сформировать строку с изображением текущего времени
        и вывести ее на экран через текстовый элемент */
        document.getElementById("clock").value =
        " "+ hours+":" + mins+":" +secs;

        setTimeout("myclock()", 1000); // Вызвать данную функцию через секунду
}
</script>
        <input type="text" size="9" id="clock">
</body></html>
```

Метод **getElementById()** рассмотрен ниже.

Объект document

Объект **document** является корнем DOM-дерева страницы, и, соответственно, через него можно получить доступ ко всем HTML-элементам, а также динамически формировать HTML-страницу на стороне клиента.

Метод **getElementById()** позволяет обратиться и (или) изменить атрибуты HTML-элемента, если задан его уникальный идентификатор (см. пример выше).

Метод **getElementsByTagName()** позволяет вернуть массив ссылок на HTML-элементы заданного типа:

```
var parr = document.getElementsByTagName('p');
alert('Содержание первого абзаца:\n'+parr[0].innerText);
```

Специальный символ **\n** осуществляет перенос каретки на новую строку (аналог тега **
**). Позволяет оформлять текст в модальных окнах и формировать удобочитаемый HTML-код (исходный, а не визуализированный).

Для динамического формирования HTML-кода страницы используются следующие методы объекта **document**: метод **open()** открывает чистый документ (если что-то было в документе, то эти данные стираются), методы **write()** и **writeln()** производят запись в новый документ HTML-элементов, а метод **close()** сообщает обозревателю, что документ сформирован и его можно отображать. Рассмотрим пример, в котором используются все указанные методы:

```
<html><head><script type="text/javascript">
function testloop() {
        var Str1 = '<hr align="center" width="' ;
        document.open();
        for (var size = 5; size <= 50; size+=5)
             document.writeln (Str1+ size+'%">');
```

```
        document.close();
}
</script></head><body>
        <input type="button" value="Сформировать новую страницу"
          onClick="testloop()">
</body></html>
```

4.8. Модели документа DHTML и DOM

Объектные модели DHTML и DOM предоставляют прямой программируемый доступ ко всем элементам HTML-документа, а совместно с событийной моделью подобный подход позволяет обозревателю Интернет обрабатывать данные о действиях пользователя, выполнять встроенные сценарии и динамически менять содержимое документа, не перезагружая его.

Модель документа DHTML

Модель документа DHTML впервые была предложена компанией Microsoft в обозревателе Internet Explorer 4 на базе идеологии, используемой в Microsoft Office Basic. Рассмотрим ту её небольшую часть, которая реализована и в других обозревателях Интернет.

В DHTML введено семейство **all**, принадлежащее объекту **document**, которое представляет собой массив всех элементов документа и позволяющее получить доступ к любому атрибуту любого HTML-элемента. При этом если в HTML-документе содержится текст, не размеченный тегами, то его уже никак нельзя изменить.

Доступ к элементам массива возможен четырьмя различными способами.

```
document.all[2] // ссылка по номеру элемента в массиве
document.all['header'] // ссылка по имени элемента, задаваемому атрибутами name и id через механизм ассоциативного массива
```

```
document.all.header // ссылка по имени элемента, задаваемому атрибутами name и id через объектный подход

var divs = document.all.tags('div'); divs[1] // ссылка через массив однотипных элементов
```

Доступ к атрибутам HTML-элементов осуществляется через их имя. Например, доступ к атрибуту **border** тега

```
<table id="tableinfo">…</table>
```

осуществляется следующим образом:

```
document.all.tableinfo.border = 0;
```

Отдельно необходимо выделить управление атрибутом **style**, которое хранит значение всех прямо определенных свойств каскадных таблиц стилей, т.е. не наследуемых для данного элемента. Доступ к свойствам возможен двумя способами:

```
(объект).style.color = 'red';

(объект).style['color'] = 'rgb(255,0,0)';
```

Для составных свойств, имеющих в названии дефис, применяются специальный алгоритм для формирования ссылок:

1) все дефисы «-» убираются из названия;
2) каждое слово, шедшее после дефиса, пишут с заглавной буквы.

Например, чтобы изменить значение свойства **border-left-color**, необходимо написать следующее:

```
объект.style.borderLeftColor = '#F00';
```

а для получения данных из свойства **text-align**

```
объект.style.textAlign
```

Рассмотрим пример того, как можно скрывать и отображать отдельные HTML-элементы:

```
<html><body><script type="text/javascript">
```

```
function changeView(object) {
        if (object.style.display=='none') object.style.display='';
        else object.style.display='none';

}
</script>
        <p>если вы наведете курсор</p>
        <p onClick="changeView(document.all.ds);"
        style="cursor: hand; color: blue"> на эту строку и 'кликните'
         ee, </p>
        <p id='ds' style="display:block;">то эта строка исчезнет</p>
        <p> а эта займет ее место<p>

</body></html>
```

Модель DHTML позволяет также полностью менять текущий HTML-элемент. Эти действия осуществляются с помощью свойств **innerHTML**, **innerText**, **outerHTML** и **outerText**.

Свойства с префиксом **inner** изменяют содержимое HTML-элемента. Свойства с префиксом **outer** переписывают HTML-элемент целиком (рис. 4.2). При этом свойства с суффиксом **Text** работают только с текстовым содержимым, из которого удалены внутренние теги разметки, тогда как свойства с суффиксом **HTML** оперируют также и с тегами.

На рис. 4.2 отмечены «области влияния» рассмотренных четырех свойств элемента на его содержимое.

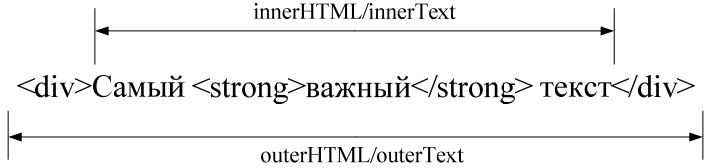


Рис. 4.2. Границы влияния свойств innerHTML, innerText, outerHTML и outerText

Для тега ** **, представленного на рис. 4.2, значением свойства **innerText** будет строка

```
Самый важный текст
```

а для **innerHTML** – строка

```
Самый <strong>важный</strong> текст
```

Значение свойства **outerText** совпадает со значением свойства **innerText**, но при его изменении теряются все теги, а **outerHTML** хранит полное описание элемента:

```
<div>Самый <strong>важный</strong> текст</div>
```

Стоит отметить: если при изменении свойства с суффиксом **Text** в строке оказываются теги, то они выводятся на экран без обработки их обозревателем.

В связи с тем, что модель DHTML реализована не полностью в обозревателях, отличных от Internet Explorer, необходимо понимать, что в ряде обозревателей вместо свойства **innerText** следует использовать свойство **textContent**. Чтобы сделать свой документ интерпретируемым, необходим следующий код:

```
if (typeof(object.innerText)!='undefined')
        object.innerText='текст';
else
        object.textContent='текст';
```

Модель документа DOM

Модель документа DOM разрабатывалась на основе DHTML. Но её коренное отличие заключается в том, что существенное значение приобретает дерево объектов, состоящее из узлов (node). Таким образом, DOM позволяет учитывать все элементы документа, в то время как DHTML может «терять» текст, который не обрамлен какими либо тегами. Более того, DOM унифицирует процесс управления тегами и, соответственно, не привязана ни к какому конкретному представлению документа (HTML, XML, SGML). Она всего лишь описывает логическую организацию документа. Ее реализация в конкретной системе представления документов ставит в соответствие узлам реальные элементы.

Рассмотрим, как осуществляется обращение к элементам и перемещение по дереву объектов на следующем примере:

```
<div>
        <h3>Заголовок</h3>
        Текст <em>курсив</em>
        <img src="img.png" alt="image">
</div>
```

На рис. 4.3 показано, как приведенный HTML-код преобразуется обозревателем Интернет к древовидной структуре. Доступ к дочерним элементам узла, назовем его, например, element, включая текстовые, осуществляется по индексу через коллекцию (массив) потомков узла **childNodes**:

```
element.childNodes[0], element.childNodes[1], …
```

В табл. 4.2 приведены некоторые свойства элементов, дающие дополнительные возможности по навигации по дереву объектов, а также позволяющие получать доступ к содержанию текстовых узлов. Данные свойства также отображены на рис. 4.3.

Таблица 4.2

Некоторые свойства узлов DOM

Свойство	Возвращаемое значение
element.nodeValue	Значение узла (содержимое текстового узла, для остальных узлов null)
childEl.parentNode	Ссылка на родителя объекта childEl
fatherEl.firstChild	Доступ к первому потомку объекта fatherEl
fatherEl.lastChild	Доступ к последнему потомку объекта fatherEl
fatherEl.previousSibling	Предыдущий братский узел
fatherEl.nextSibling	Следующий братский узел

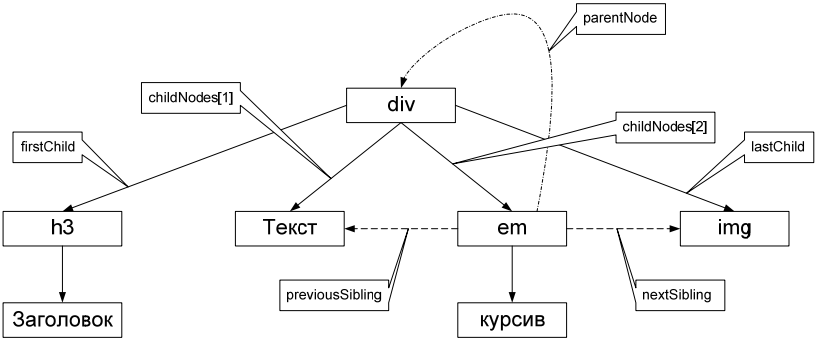


Рис. 4.3. Свойства объектов

Рассмотрим основные методы модели DOM (табл. 4.3 – 4.4), в том числе дополняющие уже рассмотренные методы объекта **document**, которые позволяют реализовать простейшие интерактивные задачи в документе: добавление и удаление HTML-элементов, изменение их содержания независимо от обозревателя Интернет.

Таблица 4.3

Дополнительные методы и объекты объекта document

Метод/свойство	Параметры	Описание
document.createElement()	Строка, в которой указывается наименование тега элемента или открывающий тег элемента с атрибутами	Создание в памяти нового HTML-элемента (не визуализируется)
document.createTextNode()	Строка, в которую вписывается визуализируемый впоследствии текст	Создание содержания, которое может быть впоследствии добавлено к любому HTML-элементу
document.body	Нет	Позволяет указать, что методы применяются только для тела HTML-документа

Таблица 4.4

Некоторые методы узлов DOM

Метод	Описание
fatherEl.appendChild(childEl)	Добавляет элемент childEl последним потомком к элементу fatherEl, возвращает добавленный элемент
element.insertBefore(childEl, brotherEl)	Вставляет элемент childEl как потомка указанного элемента сразу перед указанным потомком brotherEl, возвращает true, если операция успешна, null, если нет
fatherEl.replaceChild(newEl, oldEl)	Заменяет потомка oldEl на newEl для элемента fatherEl
fatherEl.removeChild(oldEl)	Удаляет потомка oldEl для элемента fatherEl
el.getAttribute ("style")	Возвращает значение атрибута style элемента el
el.hasAttribute ("style")	Возвращает true, если элемент el имеет атрибут style
el.setAttribute ("style", "value")	Устанавливает атрибуту с именем style значение value

Рассмотрим, как можно добавить в документ новый абзац (модель DHTML не имеет аналогичной кроссплатформенной возможности), причем неограниченное число раз:

```
<html><head><script type="text/javascript">
function addNodeP(){
        var par = document.createElement('p'); // создаем абзац
        // создаем перенос строки
        var br = document.createElement('br');
        var b = document.createElement('strong'); // создаем выделение
        // добавляем 1-й узел
        par.appendChild(document.createTextNode('1-я строка. '));
```

```
        b.appendChild(document.createTextNode('Полужирный текст'));
        par.appendChild(b); // добавляем 2-й узел
        par.appendChild(br); // добавляем 3-й узел
        // и 4-й узел
        var scndstr = '2-я строка. Просто текст';
        par.appendChild(document.createTextNode(scndstr));
        // добавляем сформированный параграф в начало документа
        document.body.insertBefore(par, document.body.firstChild);

}
</script></head><body>
        <input type="button" value="Добавить новый абзац"
        onClick="addNodeP()">
</body></html>
```

Рассмотрим, как в модели DOM будет выглядеть пример, описывающей, как скрывать и отображать отдельные HTML-элементы:

```
function changeView(object) {
        if (object.getAttribute ("style")=='display:none;')
                object.setAttribute ("style", "display:block;");
        else object.setAttribute ("style", "display:none;");

}
```

Пример выглядит похоже, однако сценарий по модели DHTML намного более безопасный. Он работает и в том случае, если для элемента определено несколько стилей. Приведенный же сценарий такую возможность не учитывает и требует добавления дополнительных проверок и функций, выделяющих нужную часть из набора стилей. Поэтому сегодня принято совмещать в одном сценарии две модели для получения наиболее безопасных сценариев.

Рассмотрим последний пример в данном разделе, подтверждающий необходимость совмещения подходов разных моделей. В этом примере мы по модели DOM заменим текущий текст абзац на другой. Если посмотреть пример выше, то можно убедиться, что

в модели DHTML эта задача в целом решается одной строкой и намного нагляднее, чем в модели DOM:

```
function replaceP(objectID){
        // создаем новый элемент-абзац
        var par = document.createElement('p');
        // присваиваем ему идентификатор заменяемого абзаца
        par.setAttribute ("id", objectID);
        // наполняем текстом
        par.appendChild(document.createTextNode('Новый текст'));
        // ищем в дереве нужный нам объект
        elem=document.getElementById(objectID);
        // заменяем старый абзац на новый
        elem.parentNode.replaceChild(par, elem);
}
```

Конечно, и в модели DOM возможна замена одной строкой, но она работает только в том случае, если у элемента всего один потомок, который является текстовым узлом, а значит не может быть признана универсальным решением:

```
document.getElementById(objectID).childNodes[0].nodeValue =
"Новый текст";
```

Задачи для самоконтроля

1. Написать универсальную функцию replaceBlock, которая заменяет текущий блочный элемент, на новый определенного типа, необязательно совпадающего с заменяемым, и поместить в него заданный текст.
2. Вставить новый абзац между двумя уже существующими.
3. После открытия нового окна визуализировать в нем всех детей тега **<body>** окна-родителя.
4. Реализовать изменение цвета заголовка каждые полсекунды, последовательно увеличивая количество красного, зеленого и синего компонентов цвета.

5. Найти готовую функцию и вставить ее в пример проверки формы (Приложение 2), которая определяет корректность ввода адреса электронной почты в форму.

Раздел 5. Серверные сценарии: языки SSI и PHP

Как было отмечено в разделе 1, для работы серверных сценариев требуется, чтобы программное обеспечение интернет-сервера поддерживало их исполнение собственными модулями или с помощью внешних интерпретаторов, работающих через интерфейс CGI. На рекомендуемой для использования в данном курсе платформе Денвер все необходимые интерпретаторы уже установлены. И здесь необходимо еще раз пояснить, что попытки загрузить файл в обозреватель Интернет его средствами или же средствами операционной системы окончатся визуализацией текстов сценариев в обозревателе, так как он не умеет их интерпретировать.

5.1. Включения на стороне сервера SSI

SSI (Server Side Includes – включения на стороне сервера) – несложный язык динамической «сборки» веб-страниц на сервере из отдельных составных частей и выдачи клиенту полученного HTML-документа. В зависимости от настроек сервера может требовать специального расширения имен файлов .shtml

Синтаксис SSI позволяет включать в текст страницы другие SSI-страницы, вызывать внешние CGI-скрипты, реализовывать условные операции (if/else), работать с переменными и т.п. Благодаря крайней простоте языка сборка SSI-страниц происходит очень быстро, однако многие возможности полноценных языков программирования в SSI отсутствуют.

SSI-директивы включаются в html-код в виде комментариев определенного формата (регистр не важен):

```
<!--#SSI-директива свойство="параметры" -->
```

Можно выделить следующие полезные сегодня директивы SSI.

Дата последней модификации файла

Полезно использовать для того, чтобы пользователь легко мог определить актуальность просматриваемых материалов:

```
<!--#flastmod file="file_name.html"-->
```

При необходимости можно регулировать формат отображения даты с помощью соответствующей директивы:

```
<!--#config timefmt="формат вывода даты" -->
```

Данная директива должна идти перед директивой, выводящей дату. Формат вывода даты можно самостоятельно найти в Интернете.

Вставка данных из внешнего файла

Значение директивы трудно переоценить. Она позволяет оформить сайт на основе шаблонов с минимальными усилиями. Зачем требуются шаблоны? Для того, чтобы создаваемый сайт смотрелся единообразно и при необходимости мог быть изменен в минимальные сроки. В отличие от каскадных таблиц стилей SSI позволяет шаблонизировать логическую структуру документа, в частности – задавать единые шапку и подвал документа, формировать единое меню.

Рассмотрим HTML-документ, подготовленный на основе данного подхода:

```
<html><body>
        <div id="header"><!--#Include Virtual="/head.html" --></div>
        <div id="menu"><!--#Include Virtual="/menu.html" --></div>
        <div id="content">Содержание</div>
        <div id="footer"><!--#Include Virtual="/foot.html" --></div>
</body></html>
```

Теги ** ** позволяют сформировать логику документа, а директива

```
<!--#Include Virtual="имя файла" -->
```

поместить в файл данные из внешних источников. Таким образом, обозревателю будет передан файл, в котором отсутствуют директивы SSI, а на их месте будут помещено содержание внешних файлов. Атрибут **virtual** позволяет задавать любые относительные пути к файлу.

Работа с переменными

SSI позволяет как управлять встроенными переменными, определяемыми сервером либо HTTP-запросом, так и создавать свои. Изменим встроенную переменную SERVER_NAME, в которой хранится адрес сервера, на котором исполняется SSI-директива (например, **www.mephi.ru**):

```
<!--#set var="SERVER_NAME" value="eai.mephi.ru" -->
```

После исполнения директивы в переменной будет новое значение **eai.mephi.ru**

Исполнение директивы можно проверить с помощью другой директивы, которая выводит в HTML-документ значение переменных:

```
<!--#echo var="SERVER_NAME" -->
```

Условный оператор

Условный оператор помогает выводить разную информацию в зависимости от значения переменных. Условный оператор обладает следующим синтаксисом:

```
<!--#if expr="УСЛОВИЕ1" -->
        HTML-код, который будет выводиться,
        если УСЛОВИЕ1 истинно
<--#elif expr="УСЛОВИЕ2" -->
        HTML-код, который будет выводиться,
        если УСЛОВИЕ1 ложно, а УСЛОВИЕ2 истинно
<--#else -->
        HTML-код, который будет выводиться,
        если все условия ложны
<--#endif -->
```

Условие – это либо строка, которая является истинной, если непустая, или набор операторов сравнения строк. Операторы могут

быть следующими: =,!=,<,<=,> и >. В случае, когда вторая строка заключена в "/"(слэши), то условие истинно, если в первой строке встречается хоть одно вхождение второй строки. Можно объединять несколько операторов сравнения с помощью операторов &&(И) и ||(ИЛИ). Для группирования условий используются "()"(скобки).

Рассмотрим пример, когда в зависимости от наличия переменной NO_BANNER на странице выводится или не выводится рекламное изображение. Сначала рассмотрим внешний файл (**adv.html**), через который подключаются необходимые директивы для отображения рекламы. Необходимо обратить внимание, что в нем действительно отсутствуют теги **<html>**, **<body>** и т.д. в связи с тем, что мы рассматриваем не отдельный самостоятельный HTML-документ, а подключаемый блок HTML-кода:

```
<!--#if expr="$NO_BANNER != 1" -->
        <p class="adv">реклама</p>
        <a href="http://www.ru" target="_blank">
        <img src="9may.gif"></a>
<!--#endif -->
```

Теперь рассмотрим файл, куда вставляется реклама (**index.html**):

```
<html><body>
        <!--#set var="NO_BANNER" value="1" -->
        <!--#Include file="adv.html" -->
        <div id="content">Содержание</div>
</body></html>
```

В файле **index.html** реклама будет отображаться только тогда, когда значение переменной **NO_BANNER** станет любым отличным от 1. Необходимо обратить внимание: если переменная используется для сравнения, то она предваряется знаком **$**. Свойство **file** гово-

рит о том, что файлы **adv.html** и **index.html** располагаются на сервере в одной директории.

5.2. Язык программирования PHP

PHP (англ. PHP: Hypertext Preprocessor – «PHP: препроцессор гипертекста», англ. Personal Home Page Tools (устар.) – «Инструменты для создания персональных веб-страниц») – скриптовый язык программирования общего назначения, интенсивно применяющийся для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинговых компаний и является одним из лидеров среди языков программирования, применяющихся для создания динамических веб-сайтов.

Популярность в области построения веб-сайтов определяется наличием большого набора встроенных средств для разработки веб-приложений. Назовем основные из них:

- автоматическое извлечение POST- и GET-параметров, а также переменных окружения веб-сервера в предопределённые массивы;
- файловые функции, успешно обрабатывающие как локальные, так и удалённые файлы;
- автоматическая отправка HTTP-заголовков, информирующих обозреватель о начале передачи HTML-документа;
- работа с cookies (текстовые данные, хранящиеся у клиента; позволяют серверу точно идентифицировать пользователя и его настройки при формировании HTML-документов) и сессиями;
- обработка файлов, загружаемых на сервер.

Рекомендуется всем HTML-документам, содержащим PHP-скрипты, давать расширение php, чтобы упростить задачу серверному программному обеспечению в идентификации содержания файла.

Синтаксис

Синтаксис PHP подобен синтаксису языка JavaScript.

Простейшая программа на PHP выглядит следующим образом:

```
<?php echo 'Hello, world!'; ?>
```

PHP исполняет код, находящийся внутри таких ограничителей, как **<?php** и **?>**. Всё, что находится вне ограничителей, выводится без изменений. В основном это используется для вставки PHP-кода в HTML-документ, например, так:

```
<html><head><title>Тестируем PHP</title></head><body>
        <?php echo '<strong>Hello, world!</strong>'; ?>
</body></html>
```

Помимо ограничителей **<?php ?>**, допускается использование сокращенных ограничителей **<?** и **?>**.

Имена переменных начинаются с символа **$**, тип переменной объявлять не нужно. В отличие от имён функций и классов, имена переменных чувствительны к регистру. Именованные константы могут быть объявлены как регистрозависимыми, так и регистронезависимыми. Переменные обрабатываются в строках, заключённых в апострофы или двойные кавычки.

PHP интерпретирует переход на новую строку в качестве пробела (так же, как HTML и другие языки со свободным форматом). Инструкции разделяются с помощью точки с запятой (**;**), за исключением некоторых случаев.

PHP поддерживает три типа комментариев: в стиле языка JavaScript: ограниченные **/* */**, начинающиеся с // и идущие до конца строки и оболочки UNIX, начинающиеся с # до конца строки.

Все операции и операторы полностью аналогичны Javascript за одним отличием: для сложения строковых переменных используется операция «**.**» (точка):

```
<?
        $gre = "He";
        $num = 3 + 2;
        while ($num < 8) {
```

```
        if ($num<=6) {$gre .= "l";}
        elseif ($num == 7) {$gre .= "o";}
        $num++;
    }
    echo '$num=' . $num . "! $gre people!\n";
?>
```

В результате работы скрипта в HTML-документе будет сформирована следующая текстовая строка:

```
$num=8! Hello people!
```

Массивы создаются также аналогично JavaScript, но без директивы New:

```
$fruit[] = 'banana';
$fruit = array('banana','papaya');
```

Встроенная функция **count()** выдает число элементов в массиве:

```
echo count($fruit);
```

Обращение к элементам массива аналогично JavaScript:

```
echo $fruit[0];
```

Шаблоны

Так же, как и SSI, PHP позволяет включать в текущий документ данные из внешних файлов. Содержащиеся в них PHP-сценарии также будут исполнены, если это допустимо. Для данной задачи используются следующие функции:

- **include("имя_файла")** – текст внешнего файла помещается в место вызова функции, после чего сразу исполняется. В случае помещения в условный оператор выполняется только при требуемом условии.
- **require("имя_файла")** – текст внешнего файла подключается заранее и определенные в нем функции доступны в любом месте основного сценария. Не используется для вставки текстовых данных.

Рассмотрим пример, где и шапка, и подвал страницы расположены во внешних файлах:

```
<? include("template/bm-head.php"); ?>
        Текст страницы
<? include("template/em.php");?>
```

Запись и чтение из файлов

Работу с файлами можно обозначить как одну из ключевых возможностей PHP, которая позволяет хранить данные между вызовами скриптов. Рассмотрим простейший пример создания файла и записи в него произвольной строки:

```
<?
$somecontent = "Поместить эту строку к файлу\n";
// Создадим новый файл для записи или обнулим ранее созданный
$handle  = fopen('test.txt', 'w');
fwrite($handle, $somecontent); // Запишем данные в файл
fclose($handle); // Закроем файл
?>
```

Функция **fopen()** открывает файл (расположенный локально или же в Интернет) для целей, которые определяются режимами, обозначенными в табл. 5.1.

Таблица 5.1

Список режимов функции fopen()

mode	Описание
'r'	Открывает файл только для чтения; помещает указатель в начало файла
'r+'	Окрывает файл для чтения и записи; помещает указатель в начало файла

Продолжение табл. 5.1

mode	Описание
'w'	Открывает файл только для записи; помещает указатель в начало файла и обрезает файл до нулевой длинны. Если файл не существует – пробует его создать
'w+'	Открывает файл для чтения и записи; помещает указатель в начало файла и обрезает файл до нулевой длинны. Если файл не существует – пробует его создать
'a'	Открывает файл только для записи; помещает указатель в конец файла. Если файл не существует – пытается его создать
'a+'	Открывает файл для чтения и записи; помещает указатель в конец файла. Если файл не существует – пытается его создать

Функция **fwrite()** записывает строку в файл.

Функция **fclose()** закрывает связь с файлом.

Рассмотрим пример чтения данных из сети Интернет:

```
<?
$fd = fopen ("http://eai.mephi.ru/", "r");
if (!fd) {exit;} // проверяем, удалось ли открыть ресурс
while (!feof ($fd)) {
        $buffer = fgets($fd, 4096);
        echo $buffer;
}
fclose ($fd);
?>
```

Функция **feof()** осуществляет проверку – достигнут ли конец читаемого файла/ресурса.

Функция **fgets()** осуществляет чтение из файла/ресурса определенные порции данных, размер которых задан (рекомендуется для

интернет-ресурса). Если величина порций не задана, то функция считывает файл построчно:

```
echo fgets($fd);
```

Обработка HTML-форм

Внутри PHP-скрипта существует несколько способов получения доступа к данным, переданным клиентом по протоколу http (из формы методами post и get).

Для обращения к переменным, переданным с помощью HTTP-запросов, используется специальный массив – **$_REQUEST**. Этот массив содержит данные, переданные методами POST и GET, а также с помощью HTTP cookies. Это суперглобальный ассоциативный массив, т.е. его значения можно получить в любом месте программы, используя в качестве ключа имя соответствующей переменной (элемента формы).

Допустим, в форме есть скрытый текстовый элемент

```
<input type=hidden name="name" value="Anna">
```

После отправки формы в вызываемом скрипте можно будет использовать переданное значение в HTML-коде следующим образом:

```
echo $_REQUEST["name"];
```

Рассмотрим по частям простейший пример вывода и обработки формы в одном файле. Функция вывод а формы:

```
<?
function display_form() {
?>

<FORM ACTION="<? echo $_SERVER['PHP_SELF']; ?>"
METHOD="post">
Имя:                    <INPUT TYPE=TEXT NAME="name"><BR>
Любимый сорт сыра:  <INPUT TYPE=RADIO NAME="cheese"
                            VALUE="md">Масдам
                    <INPUT TYPE=RADIO NAME="cheese"
                            VALUE="gau"> Гауда
```

```
                    <INPUT TYPE=RADIO NAME="cheese"
                        VALUE="ru">Российский <BR>
Когда Вы предпочитаете есть сыр:
        <INPUT TYPE=CHECKBOX  NAME="times[]"
              VALUE="m">На завтрак
        <INPUT TYPE=CHECKBOX NAME="times[]"
               VALUE="n">В обед
        <INPUT TYPE=CHECKBOX NAME="times[]"
               VALUE="d">На ужин <BR>
<INPUT TYPE=HIDDEN NAME="stage" VALUE="results">
<INPUT TYPE=SUBMIT VALUE="Рассказать">
</FORM>
<? } ?>
```

Из данного примера видно, что при большом количестве HTML-кода возможно опустить применение функции **echo** всего лишь в нужном месте закрыв php-тег. В последняя строке примера содержится закрывающая скобка для функции, т.е. форма будет выводиться только при вызове функции.

Как визуализируется форма, показано на рис. 5.1.

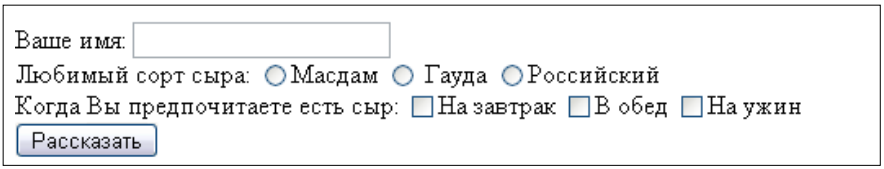


Рис. 5.1. Визуализированный вид формы

Понимая поля формы, можно сформировать условия вызова функций отображения и обработки данных формы:

```
<?
        if ($_REQUEST["stage"]) {process_form();}
        else {display_form();}
?>
```

В случае отправки формы элементу массива **$_REQUEST**["stage"] присваивается значение results (см. значение hidden-элемента формы под именем stage, это тот самый случай, когда необходимо, чтобы элемент обязательно был, но не отображался) и происходит вызов функции-обработчика формы. Если же данные в скрипт не передавались (первая загрузка), то выполняется функция вывода формы.

И, наконец, рассмотрим функцию обработки формы:

```
<?
function process_form() {

        // сократим размер имен вызываемых переменных

        $name = $_REQUEST["name"];
        $cheese = $_REQUEST["cheese"];
        $times = $_REQUEST["times"];

        // к массиву не везде можно обратиться напрямую
        $favorite_times = count($times);

        if ($cheese == 'md')
                $cheese_msg = 'Мне тоже нравится мастдам.';
            elseif ($cheese == 'gau')
                $cheese_msg = 'Гауда удивительна!';
            else $cheese_msg = 'Российский сыр - лучший.';

        if ($favorite_times <= 1)
                $times_msg = 'Можно есть сыр и чаще.';
        elseif ($favorite_times > 1 && $favorite_times < 3)
                $times_msg = 'Пора поесть.';
        else
                $times_msg = 'Но Вы едите сыр слишком часто.';

        echo "Привет $name! $cheese_msg $times_msg";
}
?>
```

Функция в зависимости от произведенного пользователем выбора формирует и выводит строку-сообщение.

Если все перечисленные выше функции будут помещены в один файл, то в итоге будет подготовлен работоспособный скрипт.

Помимо суперглобального массива **$_REQUEST** в php доступны также еще несколько суперглобальных ассоциативных массивов, позволяющих обрабатывать передаваемые клиентом данные:

- **$_GET[]** – содержит все значения, передаваемые в сценарий с помощью метода формы GET.
- **$_POST[]** – содержит все значения, передаваемые в сценарий с помощью метода формы POST.
- **$_SERVER[]** – содержит все значения, получаемые от сервера.

Необходимо отметить, что наиболее часто употребимым в скриптах элементом является **$_SERVER['PHP_SELF']**, куда помещается имя скрипта, начиная от корневой директории виртуального хоста, т.е. если строка запроса представляет собой адрес

```
http://www.mysite.ru/test/index.php?id=1&test=wet&id_theme=512
```

то элемент **$_SERVER['PHP_SELF']** будет содержать фрагмент "**/test/index.php**". Как правило, этот же фрагмент помещается в элемент **$_SERVER['SCRIPT_NAME']**.

Передача данных в php-скрипт через ссылку

При необходимости можно передавать параметры прямо в php-скрипт, минуя форму. Для этого следует сформировать следующую ссылку, которая будет передана в скрипт по методу **get**:

```
1.php?arg1=value1&arg2=value2&arr[]=foo+bar&arr[]=baz
```

где

- 1.php – название файла-скрипта;
- знак вопроса ? указывает на начало строки с данными;
- arg1=value1 – будет преобразовано в элемент массива $_REQUEST["arg1"] со значением value1;
- знак амперсанда & разделяет между собой переменные
- arr[] – позволяет сформировать массив.

Задачи для самоконтроля

1. Добавить во второй экран (после подтверждения формы) изложенного выше примера кнопку «Заполнить еще раз», при нажатии на которую будет выводиться пустая начальная форма.
2. Сделать счетчик посещений страницы. Организовать вывод этой информации на самой странице.
3. Создать три страницы на одном и том же шаблоне, в котором есть общее меню, позволяющее переключаться между страницами и в котором реализована проверка на текущую страницу, в результате чего ссылка текущей страницы блокируется.

Раздел 6. Работа с базами данных: MySQL

6.1. Основы SQL

SQL (англ. Structured Query Language – «язык структурированных запросов») – универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. SQL является, прежде всего, информационно-логическим языком, предназначенным для описания хранимых данных, для извлечения хранимых данных и для модификации данных. SQL не является языком программирования. Стандарт на язык SQL был выпущен Американским национальным институтом стандартов (ANSI) в 1986 г., а в 1987 г. Международная организация стандартов (ISO) приняла его в качестве международного. Нынешний стандарт SQL известен под названием SQL:2008.

Изначально, SQL был основным способом работы пользователя с базой данных и представлял собой небольшую совокупность команд (операторов) допускающих создание таблиц, добавление в таблицы новых записей, извлечение записей из таблиц (в соответствии с заданным условием), удаление записей и изменение структур таблиц.

В связи с усложнением язык SQL приблизился к языкам прикладного программирования, а пользователи получили возможность использовать визуальные построители запросов.

Операторы SQL делятся на:

- операторы определения данных (Data Definition Language, DDL)
- операторы манипуляции данными (Data Manipulation Language, DML)
- операторы определения доступа к данным (Data Control Language, DCL)
- операторы управления транзакциями (Transaction Control Language, TCL)

Операторы SQL не различают регистр символов, однако имена таблиц и имена баз данных могут различать регистр символов, в зависимости от используемой операционной системы.

Рассмотрим DML-операторы, которые пригодятся для выполнения рутинных операций с базой данных посредством языка PHP. Остальные операторы так или иначе реализованы через визуальные интерфейсы, требуют единичного использования. В частности один из самых популярных веб-интерфейсов к MySQL – phpMyAdmin – будет рассмотрен в п. 6.2.

Допустим у нас уже существует таблица под именем **fruits**, в которую внесены следующие значения (первая строка – имена столбцов):

id	code	name	price	country
1	7860459871032	Бананы	30	Эквадор
2	2900000984123	Апельсины	80	Марокко
3	2900000894561	Мандарины	30	Марокко
4	2900000568147	Манго	300	Тайланд

Вставка данных в таблицу

Оператор **INSERT** позволяет добавлять в таблицу данные. Общая форма:

```
INSERT into table_name (column1, column2, ...) values (value1, value2...)
```

Здесь table_name – имя таблицы, в которую надо внести данные; column1, column2 и т.д. – имена столбцов, а value1, value2 и т.д. – значения соответствующих столбцов.

Следующий оператор добавляет новую запись в таблицу **fruits**:

```
INSERT INTO fruits (code, name, price, country)
values (3598654458114, "Виноград", 120, "Франция");
```

Как и другие операторы MySQL, эту команду можно вводить на одной строке или разместить ее на нескольких строках.

Значениями для столбцов **name** и **country** являются текстовые строки, поэтому они записываются в кавычках.

Значения для **code** и **price** – числа (целые), поэтому они указываются без кавычек.

Можно заметить, что данные определены для всех столбцов кроме **id**. Значение для этого столбца задает система MySQL, которая находит в столбце наибольшее значение, увеличивает его на единицу и вставляет новое значение. Такое поведение MySQL задается в момент создания новой таблицы.

Нельзя загрузить в таблицу сразу массив данных. SQL позволяет вносить записи только построчно.

Запрос данных

Запрос данных выполняется с помощью оператора **SELECT**, который имеет следующий формат:

```
SELECT имена_столбцов FROM имя_таблицы [WHERE
...условия]
```

Часть оператора с условиями необязательна. По сути, требуется знать имена столбцов и имя таблицы, из которой извлекаются данные.

Например, чтобы извлечь штрих-коды и названия всех фруктов из таблицы **fruits**, необходимо выполнить следующую команду:

```
SELECT code, name FROM fruits
```

Чтобы вывести всю таблицу, можно либо ввести имена всех столбцов, либо воспользоваться упрощенной формой оператора **SELECT**:

```
SELECT * FROM fruits
```

Символ * в этом выражении означает 'ВСЕ столбцы'.

Рассмотрим, как ограничить вывод строк таблицы по условиям с помощью операторов сравнения = (равно) и != (не равно):

```
SELECT code, name FROM fruits WHERE country = 'Марокко'
```

В результате запроса будет выдана следующая таблица (без названий столбцов):

code	name
2900000984123	Апельсины
2900000894561	Мандарины

То есть будут выданы штрих-коды и имена только тех фруктов, страна происхождения которых – Марокко. Заметим, что слово «Марокко» в условии заключено в одиночные кавычки, что равнозначно использованию двойных кавычек. В данном случае кавычки обязательны, так как столбец country содержит текстовые значения. Также необходимо заметить: SQL не различает регистр символов. Это означает, что с равным успехом можно использовать "Марокко", "марокко" и даже "мароККо".

Рассмотрим, как выбрать фрукты, цена которых не равна 30:

```
SELECT name, price FROM fruits WHERE price!=30
```

В результате запроса будет выдана следующая таблица (без названий столбцов):

name	price
Апельсины	80
Манго	300

Аналогично применяются операторы больше (>), больше или равно (>=), меньше (<), меньше или равно (<=).

Для текста также доступен оператор поиска по шаблону **LIKE**.

Например, необходимо найти все фрукты, начинающиеся на букву **М**. Для этого необходимо использовать инструкцию

```
SELECT name, country FROM fruits WHERE name LIKE "М%"
```

В результате запроса будет выдана следующая таблица (без названий столбцов):

name	country
Мандарины	Марокко
Манго	Тайланд

Знак **%** действует как символ-заместитель (аналогично использованию * в системах DOS и Linux). Он заменяет собой любую последовательность символов. Таким образом, "М%" обозначает все строки, начинающиеся с буквы **М**. Аналогично "%м" выбирает строки, которые заканчиваются символом **м**, а "%м%" – строки, содержащие букву **м**.

Удаление записей из таблицы

Для удаления записей из таблицы используется оператор **DELETE**, требующий задания имени таблицы и необязательных условий:

```
DELETE from имя_таблицы [WHERE условия];
```

Если никакие условия не будут заданы, то удаляются все данные в таблице. Тем не менее в большинстве случаев требуется удалить какую-либо определенную запись или ряд записей, обладающих единым признаком. В нашем случае предположим, что в результате успешных продаж манго кончилось, и, как следствие, его требуется удалить из списка доступных фруктов:

```
DELETE from fruits WHERE name = 'Манго';
```

Обновление записей в таблице

Для обновления записей в таблице используется оператор **UPDATE**, требующий задания имени таблицы и необязательных условий:

```
UPDATE имя_таблицы
SET <присваивание1 [, присваивание2, ...]>
[WHERE <условие>]
```

Допустим, у нас изменились цены на мандарины. В этом случае обновить их в таблице мы можем следующим запросом:

```
UPDATE fruits SET price="50" WHERE name="Мандарины"
```

6.2. Управление базой данных через phpMyAdmin

phpMyAdmin – веб-приложение с открытым кодом, написанное на языке PHP, представляющее собой веб-интерфейс для администрирования СУБД MySQL и позволяющее через обозреватель Интернет осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных. Приложение пользуется большой популярностью у веб-разработчиков, так как позволяет управлять СУБД MySQL без не-

посредственного ввода SQL команд, предоставляя дружественный интерфейс.

На сегодняшний день phpMyAdmin широко применяется на практике. Так, подавляющее большинство российских хостеров используют это приложение в качестве панели управления для того, чтобы предоставить своим клиентам возможность администрирования выделенных им баз данных.

В данный момент phpMyAdmin позволяет:

- создавать и удалять базы данных;
- создавать, копировать, удалять, переименовывать и изменять таблицы;
- осуществлять сопровождение таблиц;
- удалять, править и добавлять поля;
- выполнять SQL-запросы, в том числе пакетные;
- управлять ключами;
- загружать текстовые файлы в таблицы;
- создавать графическую схему базы данных в формате PDF;
- осуществлять поиск в базе данных или в её разделах.

В Денвере phpMyAdmin располагается по адресу http://localhost/Tools/phpMyAdmin

Создание базы данных

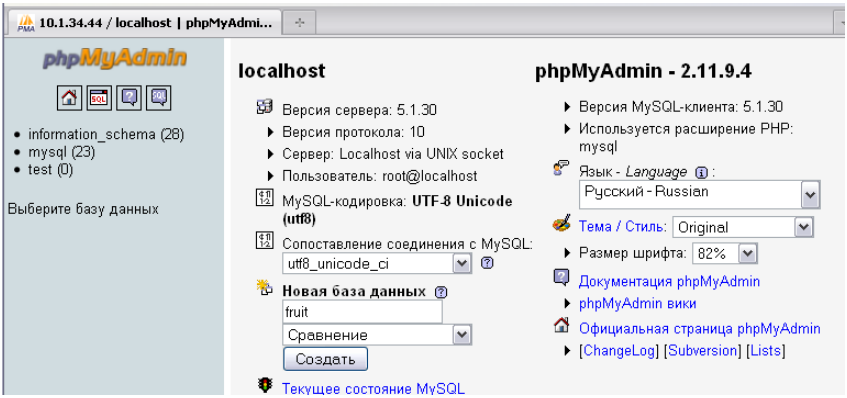


Рис. 6.1. Главная страница phpMyAdmin

На главной странице phpMyAdmin найдите строку «Новая база данных» (рис. 6.1). В поле под надписью наберите название базы данных и нажмите кнопку «Создать». На новой странице будет указано – успешно или неуспешно проведена процедура (рис. 6.2).

Создание таблицы

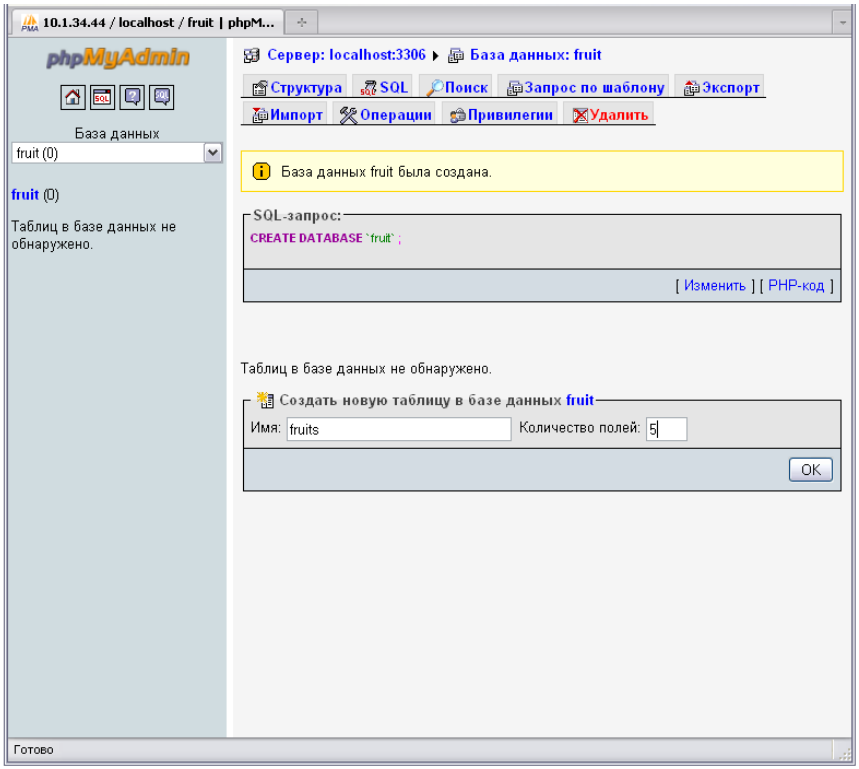


Рис. 6.2. Создание таблицы в новой базе данных

После создания базы данных тут же возможно создать в ней первую таблицу (рис. 6.2), задав ее имя и количество столбцов. Опираясь на рассмотренный выше пример, зададим пять столбцов. И в следующем экране phpMyAdmin предложит описать их свойства (рис. 6.3).

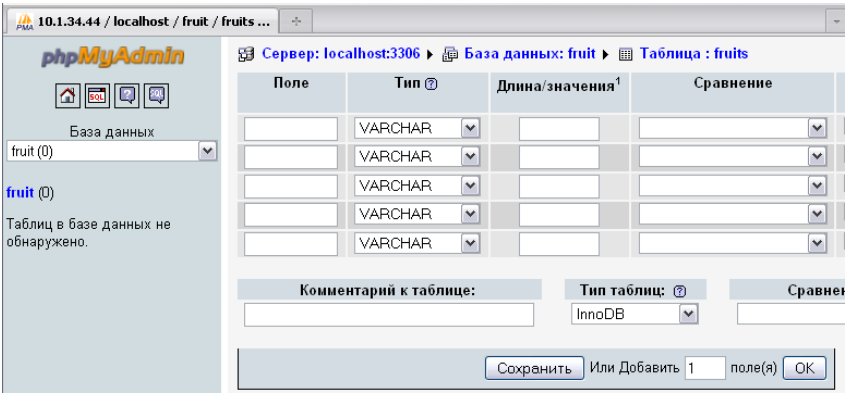


Рис. 6.3. Диалог описания столбцов

Заполним необходимую информацию о столбцах согласно таблице на с. 118 с учетом смысла полей (табл. 6.1).

Таблица 6.1

Определение параметров столбов в phpMyAdmin

Поле	**Тип**	**Длина/ значение**	**Дополнительно**		Примечание
id	TINYINT		auto_increment	+	TINYINT позволяет описать до 255 записей
code	BIGINT				BIGINT позволяет хранить штрихкоды, состоящие из 13 цифр.
name	VARCHAR	50			VARCHAR позволяет хранить строку. При этом мы определяем, что такая строка не должна быть больше 50 символов, что более чем достаточно для любых названий фруктов
price	FLOAT				FLOAT позволяет хранить дробные числа (копейки)
country	VARCHAR	50			Аналогично полю name

Изначально поле **id** планировалось для того, чтобы каждая запись смогла быть уникальной. Очевидно, что наименований фруктов и стран, где они растут, не слишком большое количество, поэтому полю присвоен тип TINYINT. Параметр «auto_increment» задает, что при добавлении новых записей в рассматриваемое поле должно попадать значение последнего находящегося в таблице **id**, увеличенное на единицу. Плюс в столбце  (флаг в phpMyAdmin) запрещает появление в столбце одинаковых значений, чем обеспечивается уникальность записи и возможность обработки именно её данных.

Также полю «Сравнение» присвоим значение «cp1251_bin», чтобы MySQL знал, в какой кодировке мы будем загружать буквы национального (кириллического) алфавита. После того как нажата кнопка «Сохранить», в таблицу будут добавлены столбцы. Результат добавления отображен на рис. 6.4. В данный экран можно попасть в любое время и внести изменения в любой из столбцов.

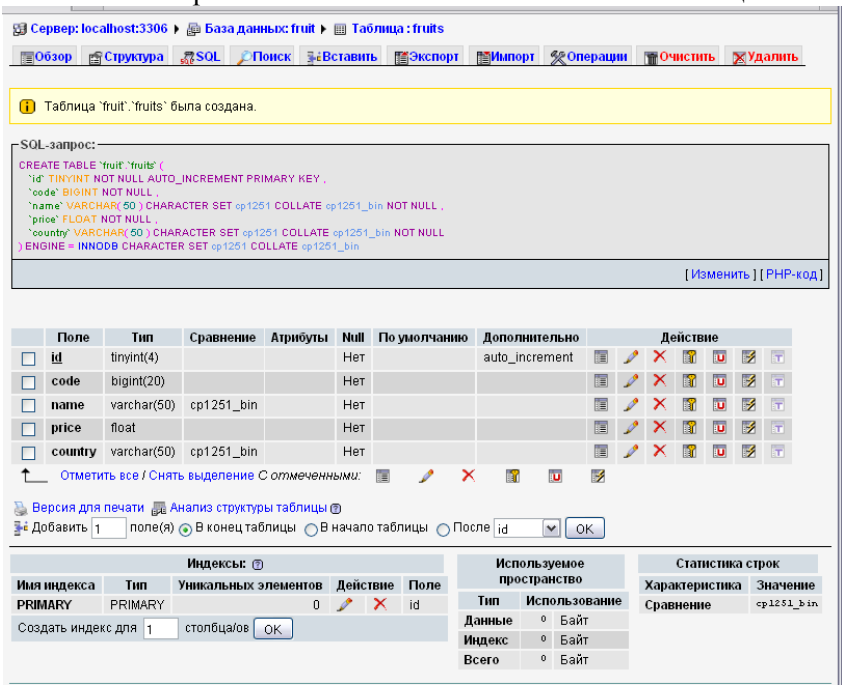


Рис. 6.4. Результат добавления столбцов в таблицу

Вставка данных

Нажав в меню на кнопку «Вставить», можно попасть в экран, представленный на рис. 6.5. Введя данные, осуществим предварительное наполнение таблицы.

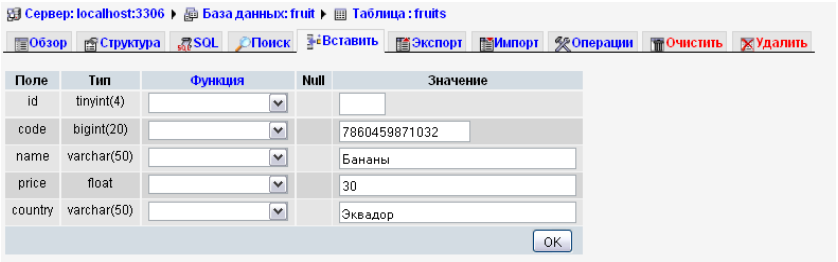


Рис. 6.5. Вставка данных в таблицу средствами phpMyAdmin

После того как база данных и таблица создана, можно приступать к созданию своей системы управления через php-скрипты.

6.3. Создание собственного PHP-скрипта для управления базой данных

Прежде чем начать извлекать или загружать какие-либо данные, необходимо подключиться к базе данных. Для этого существуют следующие функции:

- **mysql_connect** (*имя_сервера*, *имя_пользователя*, *пароль*) – открывает соединение с сервером, все параметры необязательны и зависят от настроек сервера;
- **mysql_select_db** (*имя_базы, указатель_на_соединение*) – выбирает базу данных, указатель необязателен, если внутри скрипта происходит работа только с одной базой данных.

Таким образом, начало нашего скрипта (опуская стандартные html-теги) будет выглядеть следующим образом с учетом того, что пароль для пользователя root в Денвере не определен, а сама база данных располагается на сервере localhost:

```
mysql_select_db("fruit", mysql_connect("localhost", "root"));
```

Ранее в разделе 6.2. мы уже создали базу данных и знаем все ее поля. Исходя из этого, рассмотрим код отображения формы, которая позволит нам как добавлять новые значения в таблицу, так и редактировать уже загруженные:

```
<?
function display_form () {
        // Надпись на кнопке отправки формы по (умолчанию)
        $submit_msg = "Добавить";
?>
<form method="post" action="<? echo $_SERVER['PHP_SELF'] ?>">
<?
        if ($_REQUEST["id"]) {
                $sql = "SELECT * FROM fruits WHERE
                        id=".$_REQUEST["id"];
                $tr = mysql_fetch_array(mysql_query($sql));
?>
        <input type=hidden name="id" value="<? echo $tr["id"] ?>">
<?
                // Надпись на кнопке отправки формы (новая)
                $submit_msg = "Изменить";
} // конец if
?>
        Название фрукта: <input type="Text" name="name"
                value="<? echo $tr["name"]; ?>"><br>
        Штрих-код: <input type="Text" name="code"
                value="<? echo $tr["code"] ?>"><br>
        Цена: <input type="Text" name="price"
                value="<? echo $tr["price"]; ?>"><br>
        Страна происхождения: <input type="Text" name="country"
                value="<? echo $tr["country"]; ?>"><br>
        <input type="submit" name="submit"
                value="<? echo $submit_msg; ?>">
</form>
```

```
<?  } // конец функции display_form() ?>
```

В приведенном коде нам знакомы практически все конструкции за исключением следующих:

- **mysql_query**(*SQL-запрос, указатель_на_базу_данных*) – возвращает результат обработки базой данных SQL-запроса;
- **mysql_fetch_array**(*результат_SQL-запроса*) – обрабатывает результата запроса, возвращая ассоциативный массив.

В случае, если в скрипт передается уникальный идентификатор записи **id**, производится получение данных этой записи в ассоциативный массив **$tr**, элементы которого затем используются при формировании формы. Как мы увидим дальше, уникальный идентификатор поступает в скрипт в двух случаях: при изменении либо при удалении элемента. При операции изменения (до и после) данные будут выводиться в форму, а при удалении ничего не будет выводиться, так как вывод формы (см. следующий блок кода) идет после операции удаления и на запрос по несуществующему к тому моменту уникальному идентификатору база выдает пустой ответ:

```
if ($_REQUEST["submit"]) {change_db();}
print_list();
display_form();
```

В приведенном листинге мы видим основное тело программы. В случае отправки данных формы или удаления в скрипт передается переменная submit, которая инициирует вызов функции **change_db()**, производящей изменения в базе данных. Функция **print_list()** выводит текущее содержание изучаемой нами таблицы и формирует управляющие ссылки для операций изменения и удаления записей.

Рассмотрим функцию **change_db()**:

```
<?
function change_db() {
/* Функция будет выполняться, только если все поля формы заполнены
или планируется удаление записи */
        if (($_REQUEST['name']
                && $_REQUEST['code']
                        && $_REQUEST['price']
                                && $_REQUEST['country']))
                || ($_REQUEST["submit"] == 'delete')){
                // Введение сокращенных имен
                $name = $_REQUEST['name'];
                // для переданных данных. Если планируется
                // удаление, то переменные будут пустыми
                $code = $_REQUEST['code'];
                $price = $_REQUEST['price'];
                $country = $_REQUEST['country'];
                if (!$_REQUEST["id"]) {
                // id нет только если добавляется новая запись
        $sql = "INSERT INTO fruits (name, code, price, country)
                        VALUES ('$name', $code, $price, '$country')";
        $info_msg = "Запись добавлена";
                } else if ($_REQUEST["submit"] == 'delete') {
                // обработка вызова на удаление
        $sql = "DELETE FROM fruits
                        WHERE id=".$_REQUEST['id'];
        $info_msg = "Запись N" . $_REQUEST['id'] . " удалена";
                } else {
                // в остальных случаях подразумевается
                // изменение записи
```

```
        $sql = "UPDATE fruits SET name='$name', code=$code,
                        price=$price, country='$country'
                        WHERE id=".$_REQUEST['id'];
        $info_msg = "Запись изменена";

                }

                mysql_query($sql);

                echo '<p style="background-color:#FF9999;
                        border: red 1px solid; padding:2px 6px;">' .
                         $info_msg . "</p>";

        } // конец перового if
} // конец функции change_db() ?>
```

Как можно заметить, в зависимости от входящих условий формируется SQL-запрос и информационное сообщение, которые потом соответственно выполняются и выводятся пользователю.

И наконец, рассмотрим функцию вывода данных из таблицы **print_list()**:

```
<? function print_list(){

        echo "<table border=1>
                <tr><th>id</th><th>code</th><th>name</th>
                <th>price</th><th>country</th>
                <th colspan=2>Редактирование</th></tr>";

        $result = mysql_query("SELECT * FROM fruits");

        while ($tr = mysql_fetch_array($result)) {

                printf("<tr><td>%s</td><td>%s</td><td>%s</td>
                        <td>%s</td><td>%s</td>", $tr['id'], $tr['code'],
                        $tr['name'], $tr['price'], $tr['country']);

                printf("<td><a href=\"%s?id=%s\">(изменить)</a>
                        </td>", $_SERVER['PHP_SELF'], $tr['id']);
```

```
                printf("<td><a href=\"%s?id=%s&submit=delete\">
                        (удалить)</a></td></tr>",
                        $_SERVER['PHP_SELF'], $tr["id"]);
        }
        echo "</table>";
        echo '<a href="' . $_SERVER['PHP_SELF'] . '">
                Новая запись</a>';
} // конец функции print_list()
?>
```

В приведенном коде первый оператор **echo** формирует заголовок таблицы. Второй извлекает все данные из таблицы, а третий в цикле формирует строки таблицы. После чего таблица закрывается и добавляется строка, позволяющая создать новую запись (загрузить пустую форму). Здесь мы видим интересную функцию **printf**(*строка-шаблон, переменные*), которая выводит данные в соответствии с заранее заданным шаблоном (%s в строке заменяется на переменную в соответствии с порядком упоминания), что позволяет существенно упрощать описание вывода данных в HTML-документ.

Пример визуализации скрипта приведен на рис. 6.6.

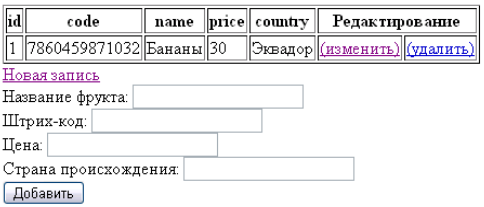


Рис. 6.6. Результат визуализации работы скрипта управления базой данных

Если все перечисленные выше функции будут помещены в один файл, то в итоге будет подготовлен работоспособный скрипт.

Задачи для самоконтроля

1. В приведенном выше примере добавить PHP-контроль за вводом данных в форму и вывод сообщений о том, что следует дозаполнить в форме (все поля обязательны), а также реализовать проверку на корректность данных (в штрихкоде 13 цифр, цена – только цифры).
2. В приведенном выше примере добавить форму, позволяющую накладывать фильтр на каждое поле таблицы, т.е., например, выводить только строки, в которых цена фрукта – 80 рублей, и только те столбцы, которые определил пользователь.

Раздел 7. Готовые веб-системы

Интернет-технологии становятся всё более зрелыми и сегодня доступно достаточно большое количество интернет-систем с многолетней историей и обширным функционалом. Данное обстоятельство позволяет не создавать систему с нуля, а адаптировать (зачастую только изменив внешний вид) под свои нужды существующую. Рассмотрим наиболее ярких представителей указанных систем, работающих на базе рассмотренных в настоящем пособии технологий.

7.1. Форумы

phpBB

http://www.phpbb.com/

phpBB (PHP Bulleting Board) – популярный бесплатный веб-форум с открытым исходным кодом, разработанный на скриптовом языке PHP, поддерживающий различные СУБД, включая MySQL, PostgreSQL, MS SQL Server, MS Access, а также Oracle.

Кроме поддержки различных СУБД несомненными достоинствами phpBB являются:

- несложная в использовании система шаблонов;
- многоязычный интерфейс: языковые файлы переведены на более чем 50 языков и доступны для свободного скачивания с официального сайта phpBB;
- большое сообщество пользователей, готовых прийти на помощь;
- большое количество доступных и обновляемых модификаций.

Invision Power Board

http://www.ibresource.ru/

Invision Power Board (сокращенно – IPB, IP.Board или IP Board) представляет собой один из самых популярных в мире решений для разворачивания форумов. Данное программое обеспечение разрабатывается фирмой Invision Power Services, Inc., написано полностью на PHP и использует для ведения своей базы данных сервер

MySQL (дополнительно имеется поддержка других СУБД, таких, например, как Microsoft SQL Server и Oracle).

Несмотря на то, что Invision Power Board является коммерческим продуктом, сообщества, занимающиеся его поддержкой и модификацией, довольно популярны и насчитывают тысячи человек по всему миру. Бóльшая часть модификаций и стилей оформления, разрабатываемых данными сообществами, бесплатна и свободна для загрузки.

7.2. Блоги

Wordpress

http://ru.wordpress.org/

На сегодняшний день WordPress – самая популярная система для ведения блогов, обладающая следующими возможностями:

- публикация с помощью сторонних программ и сервисов;
- моментальная публикация;
- простота установки, настройки;
- поддержка веб-стандартов (XHTML, CSS);
- поддержка RSS, Atom, trackback, pingback;
- подключаемые модули (плагины) с уникальной не сложной системой их взаимодействия с кодом;
- поддержка так называемых тем, позволяющих легко менять как внешний вид, так и способы вывода данных;
- «темы» реализованы как наборы файлов-шаблонов на PHP, что положительно сказывается на скорости и гибкости;
- громадные библиотеки «тем» и «плагинов»;
- заложенный потенциал архитектуры позволяет легко реализовывать сложные решения;
- наличие ЧПУ (человекопонятный URL).

bBlog

http://www.bblog.com/

Простейшая система ведения блогов. Содержит:

- визуальный WYSYWIG редактор;

- добавление новостей с возможностью указать теги, дату разрешить/запретить комментарии;
- форма обратной связи с CAPTCHA;
- комментарии с защитой от спамботов;
- удобная и интуитивно понятная панель администратора.

7.3. Фотогалереи

Coppermine Photo Gallery

http://coppermine-gallery.net/

Многофункциональная фотогалерея, по функционалу сравнимая с форумом. Основные особенности:

- загрузка изображений через ftp;
- категоризация изображений и возможность создания любой иерархии;
- автоматическое создание миниатюр и страниц с миниатюрами, разбитые по количеству отображаемых миниатюр;
- многоуровневая регламентация доступа, включая квотирования места под галерею, паролирования альбомов и отдельных фотографий, премодерирование и т.д.;
- исключение доступа к изображений по прямому адресу.

Gallery

http://gallery.menalto.com/

Gallery Project – открытый проект для управления и публикации цифровых фотографий и видеоклипов посредством публикации их на веб-сервере. Фотографии можно изменять в размере, поворачивать, накладывать и включать в иерархическую структуру, управлять которой можно посредством развитых средств управления пользователю, наделённому полномочиями.

jsImageBox

http://jsimagebox.ru

jsImageBox – компактный и несложный в подключении скрипт, который позволяет показывать увеличенные изображения во всплывающем блоке. Код скрипта занимает всего 9 кбайт. Для его

работы на страницу нужно подключить только один javascript-файл. Скрипт написан без использования сторонних фреймворков и библиотек, поэтому не должен вызвать конфликтов при подключении на сайты, уже сделанные с использованием какого-либо фреймворка.

7.4. Каталоги ссылок

LinkExchanger

http://linkexchanger.ru/

Основные возможности:

- конфигурация скрипта через интерфейс администратора;
- проверка ответных ссылок;
- статические (постоянные) ссылки;
- отправка сообщений на e-mail;
- поиск по каталогу;
- резервное копирование;
- автообновления до новой версии.

Распространяется свободно.

FairLinks

http://www.fairground.ru/fairlinks/

Основные возможности:

- интеграция в любой сайт и дизайн при наличии php и MySQL;
- использование ЧПУ – «человекопонятный URL»;
- создание каталога с очень жёсткими правилами и «белого» каталога;
- автопроверка обратных ссылок;
- малая нагрузка на сервер;
- управление действиями путём указания сценариев;
- способность устанавливать в одну базу MySQL неограниченное количество копий;
- создание на одном сайте неограниченного количества как копий, так и самостоятельных разных каталогов ссылок;

- правка пользователями своих ссылок;
- работа в полностью автоматическом режиме.

Распространяется свободно.

7.5. Системы управления документами и файловые архивы

KnowledgeTree

http://www.knowledgetree.com

Система управления документами KnowledgeTree позволяет управлять знаниями, контролировать версии документов, управлять иерархиями документов. Система поддерживает распространенные файловые форматы (MS Word, MS Excel, PDF, TXT, HTML), расширенные метаданные, позволяет создавать собственные типы документов, программно контролируемые ссылки из документов, которые обеспечивают целостность данных и избавляют от необходимости постоянно пересылать новые версии. При этом необходимо отметить, что процедура публикации документа достаточно проста. Поддерживаются агенты подписки. Производительность системы можно поднять с помощью архивирования в соответствии с датами и временем окончания срока действия или частотой обращения к документам.

Распространяется свободно.

RW:Download

http://www.rwscripts.com

Скрипт, представляющий собой каталог файлов (файловый архив) с богатым набором функций. Среди них:

- управление учётными записями пользователей с возможностью распределения по разным группам, имеющим разные права;
- добавление описания и рейтинга к каждому файлу;
- добавление комментариев, рейтинга от пользователей;
- защита от личеров;
- галерея файлов;
- интеграция с Invision Power Board или Vbulletin;
- поиск по содержимому;

- панель управления для пользователей и много другое.

Распространяется свободно.

7.6. Статистика

CNStats

http://www.cn-software.com/ru/cnstats/

CNStats – это гибкая и универсальная коммерческая система для сбора и анализа статистики посещаемости сайта, а также абсолютно точная статистика сайта, которая учитывает все посещения. Более 40 базовых отчетов доступны через веб-интерфейс. CNStats работает на любом хостинге, который поддерживает PHP и MySQL.

CNStats быстро и просто устанавливается на веб-сайт, умеет считать как посетителей, так и роботов. Поддерживает различные варианты счетчиков. Уникальная карта мира, построенная на базе GeoIP, контроль посетителей он-лайн, накладываемые многоуровневые фильтры, счетчик посещений, сводный отчет по e-mail.

TrackSite

http://tracksite.ru/

Основные особенности:

- собственная независимая платформа веб-аналитики;
- аналитика в режиме реального времени;
- конверсия и сегментация аудитории сайта;
- настраиваемые интерактивные отчёты;
- визуализация сложных данных;
- веб-аналитика для бизнеса;
- анализ поисковых запросов;
- анализ рекламных кампаний;
- анализ аудитории посетителей;
- анализ аудитории Atom/RSS потоков;
- анализ загрузок файлов;
- метрика веб-сервера;
- наблюдение активности поисковых роботов.

Коммерческая система.

7.7. Интернет-магазины

OsCommerce

http://www.oscommerce.com/

OsCommerce Online Merchant является программой интернет-магазина с открытым исходным кодом для электронной коммерции, которая доступна для свободного распространения под лицензией GNU (General Public License). Она имеет богатый функционал, что позволяет владельцам магазинов установить, запустить и поддерживать интернет-магазины с минимальными усилиями и без каких-либо расходов, сборов или ограничений.

Вокруг osCommerce сформировалось огромное сообщество (более 140 000 участников), благодаря которому существует более 4 000 контрибуций (различных модулей для osCommerce), позволяющих изменять и дополнять функции магазина самым разным образом. По всему миру функционируют десятки тысяч магазинов на базе osCommerce (на дату написания настоящего пособия более 248 000 магазинов официально зарегистрировано в каталоге сайта поддержки системы).

PrestaShop

http://prestadev.ru/

Интернет-магазин, созданный с помощью PrestaShop, обладает всем необходимым функционалом для начала продаж через Интернет. Перечислим лишь некоторые интересные особенности:

для пользователей

- специальное предложения (скидки, подарочные ваучеры);
- сопутствующие товары (аксессуары);
- возможность предварительного заказа товара, которого еще нет в магазине;
- множество методов оплаты;
- поиск;
- возврат товара и отпуск товара в кредит по спискам;
- отслеживание доставки;
- список предпочтительных товаров (Wishlist);
- программа поддержки лояльных клиентов;
- программа поддержки ваших покупок другом;

для администрации

- уменьшающиеся цены (скидка на количество);
- отправка SMS;
- штрих-коды;
- контекстная помощь;
- доставка статуса уведомлений по электронной почте;
- режим отключения магазина;
- минимальная сумма заказа;
- поиск аналогов;
- поддержка SSL (Secure Sockets Layer) шифрования.

Распространяется свободно.

7.8. Аукционы

PHP Pro Bid

http://www.phpprobid.com

PHP Pro Bid – простая в управлении, но очень мощная система для организации аукционов. Идеально подходит для небольшого бизнеса. Инсталляция занимает менее 10 мин, после чего вы становитесь полноправным владельцем собственного аукциона типа e-Bay. Скрипт обладает хорошей панелью администратора, которая дает полный контроль над системой.

Web Auction

http://apps.weblite.ca/product_catalog

Персональный аукцион, позволяющий быстро и просто запустить собственный аукцион.

7.9. Wiki

Вики характеризуется такими признаками, как:

- возможность многократно править текст посредством самой вики-среды (сайта), без применения особых приспособлений на стороне редактора:
  - особый язык разметки – так называемая вики-разметка, которая позволяет легко и быстро размечать в тексте структурные элементы и ги-

перссылки; форматировать и оформлять отдельные элементы;
    - учёт изменений (версий) страниц: возможность сравнения редакций и восстановления ранних;
- появление изменений сразу после их внесения;
- разделение содержимого на именованные страницы;
- гипертекстовость (связь страниц и подразделов сайта через контекстные гиперссылки);
- множество авторов. Некоторые вики могут править все посетители сайта.

Самые известные Wiki-системы:

- MediaWiki (http://www.mediawiki.org/)
- DokuWiki (http://wiki.splitbrain.org/wiki:dokuwiki)

Распространяется свободно.

7.10. Системы управления контентом

Системы управления контентом интегрируют в себя все вышеуказанные программные системы, позволяя создавать многофункциональные интерактивные порталы.

Joomla!

http://www.joomla.org/

Joomla! – система управления содержимым, написанная на языках PHP и JavaScript, использующая в качестве хранилища базу данных MySQL. Является свободным программным обеспечением, распространяемым по лицензии GNU GPL.

CMS Joomla! включает в себя различные инструменты для изготовления веб-сайта. Важной особенностью системы является минимальный набор инструментов при начальной установке, который дополняется по мере необходимости. Это снижает загромождение административной панели ненужными элементами, а также нагрузку на сервер и экономит место на хостинге.

Основные возможности:

- функциональность можно расширять с помощью дополнительных модулей (расширений, плагинов);
- модуль безопасности для многоуровневой аутентификации пользователей и администраторов;

- система шаблонов позволяет легко изменять внешний вид сайта;
- настраиваемые схемы расположения модулей, включая левый, правый и центральный блоки меню;
- к преимуществам системы можно отнести то, что все модули, компоненты, плагины, шаблоны можно написать самому, разместить их в структурированном каталоге расширений или отредактировать существующее расширение по своему усмотрению.

Возможности администрирования:

- для каждой динамической страницы можно создать свои описание и ключевые слова в целях повышения рейтинга в поисковых системах;
- начало и окончание публикации любых материалов можно запрограммировать по календарю;
- ограничение доступа к определённым разделам сайта только для зарегистрированных пользователей;
- настраиваемые схемы расположения элементов по областям шаблона;
- различные модули (последние новости, счётчик посещений, подробная статистика посещений, гостевая книга, форум и другие);
- создание не одной, а нескольких форм обратной связи для каждого контакта;
- модуль приёма от удалённых авторов новостей, статей и ссылок;
- менеджер рассылки новостей. Поддержка более чем 360 служб рассылки новостей по всему миру;
- Ввстроенный визуальный редактор TinyMCE;
- ЧПУ (человекопонятный URL);
- около 4000 готовых модулей и компонентов.

Распространяется свободно.

Drupal

Drupal – система управления сайтом (CMS), написанная на языке PHP и использующая в качестве хранилища данных реляционную базу данных (поддерживаются MySQL, PostgreSQL и др.).

Drupal является свободным программным обеспечением, защищённым лицензией GPL, и развивается усилиями энтузиастов со всего мира.

Архитектура Drupal позволяет применять его для построения различных типов сайтов – от блогов и форумов до информационных архивов или сайтов новостей. Функциональность обеспечивается подключаемыми модулями, обращающимися к общему API Drupal. Стандартный набор модулей включает такие функции, как новостная лента, блог, форум, загрузка файлов, сборщик новостей, голосования, поиск и другие. Большое количество дополнительных модулей, значительно расширяющих базовые функции, можно взять на официальном сайте.

Наиболее важные функции, предоставляемые модулями, входящими в поставку Drupal:

- единая категоризация всех видов содержимого (таксономия) – от форумных сообщений до блогов и новостных статей;
- широкий набор свойств при построении рубрикаторов: плоские списки, иерархии, иерархии с общими предками, синонимы, родственные категории;
- вложенность категорий любой глубины;
- поиск по содержимому сайта, в том числе по таксономии и пользователям;
- разграничение доступа пользователей к документам (ролевая модель);
- динамическое построение меню;
- поддержка XML-форматов:
    - вывод документов в RDF/RSS,
    - агрегация материалов с других сайтов,
    - BlogAPI для публикации материалов с помощью внешних приложений;
- авторизация через OpenID;
- ЧПУ (человекопонятный URL);
- переводы интерфейса сайта на разные языки, а также поддержка ведения разноязычного контента;

- возможность создания сайтов с пересекающимся содержимым (например, общей базой пользователей или общими настройками);
- раздельные конфигурации сайта для различных виртуальных хостов (в том числе собственные наборы модулей и тем оформления для каждого подсайта);
- механизм для ограничения нагрузки на сайт (автоматическое отключение при высокой посещаемости части информационных блоков и модулей);
- уведомления о выходящих обновлениях модулей.

Распространяется свободно.

1С-Битрикс: Управление сайтом

http://www.1c-bitrix.ru/products/cms/

Система в различных редакциях может удовлетворить самого взыскательного заказчика.

Управление контентом (CMS):

- статическая информация – тексты, изображения, таблицы и т.п.;
- динамическая информация – ленты новостей, каталоги товаров, статьи;
- визуальный редактор;
- документооборот;
- SEO-оптимизация;
- фотогалереи, массовая загрузка фотографий;
- медиа-плеер (видео и аудио);
- управление рекламой на сайте;
- распределение прав доступа к сайту;
- поиск по сайту с соблюдением прав доступа, морфологический анализ.

Интернет-магазин:

- полное управление продажами с сайта;
- розничные, оптовые и дилерские цены;
- партнерские и аффилиатские сети;
- полная интеграция с «1С:Предприятие 8»;

- продажа электронного контента;
- интеграция с основными платежными системами;
- автоматический расчет стоимости доставки;
- скидки и наценки;
- профайлы покупателей;
- настройка корзины;
- мастер оформления заказа;
- разделение ролей по управлению магазином.

Коммуникации:

- форумы, закрытые группы, звания, рейтинг участников, антимат;
- блоги, календарь сообщений, облако тегов;
- служба техподдержки;
- веб-формы, заявки, анкеты;
- опросы и голосования;
- почта;
- подписка и рассылки;
- обучение и тестирование;
- социальная сеть – сообщества, рабочие группы, друзья;
- бизнес-процессы.

Безопасность:

- проактивная защита.

Веб-аналитика:

- статистика посещаемости сайта;
- география по странам;
- ссылающиеся сайты;
- анализ эффективности рекламных кампаний, ROI, событийный анализ;
- он-лайн-монитор;
- индексация сайта поисковиками.

Сервисы:

- веб-кластер;

- социальные сервисы;
- монитор производительности;
- интеграция с Active Directory и LDAP;
- веб-сервисы;
- компрессия при передаче данных.

Технологические особенности:

- технология обновлений SiteUpdate;
- производительность;
- безопасность;
- многосайтовость;
- адаптивный интерфейс;
- среда для разработки индивидуальных решений (FrameWork);
- открытые исходные тексты;
- поддержка UNIX и Windows:
- поддержка БД MySQL, Oracle, MSSQL Коммерческая система.

Коммерческая система.

7.11. Корпоративные порталы

Корпоративные порталы – это подмножество систем управления контентом, имеющих целевой функционал, ориентированный на использование внутри организации.

LifeRay

http://www.liferay.com

LifeRay Portal – это корпоративный портал, позволяющий управлять корпоративным контентом (web-контент, документы, мультимедиа файлы) и организовывать совместную работу (для этого в составе решения есть календарь, вики, форумы, блоги, мессенджер). Интерфейс создается с помощью визуального конструктора, блоки (портлеты) можно перетаскивать мышкой или настраивать с помощью простой менюшки, без необходимости разбираться в CSS и программном коде. LifeRay написан на Java, что обеспечивает высокий уровень безопасности и поддержку стандартных (для

корпоративных порталов) java-портлетов. Его можно настраивать как угодно под индивидуальные портебности компании, разрабатывать собственные модули, интегрировать с другими бизнес-приложениями. По умолчанию LifeRay поддерживает гаджеты Google и позволяет работать с ним прямо из приложений MS Office. Enterprise-версия распространяется по подписке (за определенную месячную плату) и, в отличие от бесплатной, предлагает расширенную функциональность, а также услуги по настройке и поддержке решения.

Система LifeRay занимает одно из главных мест среди java-порталов и является ведущей корпоративной портальной базой с открытыми исходными текстами, предлагающей интегрированные Web-публикации и управление контентом.

Прогрессивная лицензия класса Open Source, полный набор средств обеспечения и услуг и интеграция с современными технологиями представляют собой лучшую комбинацию производительности и качества в портальной индустрии на сегодняшний день.

Функциональные возможности системы LifeRay не уступают коммерческим портальным системам, но, в отличие от последних, LifeRay не требует покупки дорогостоящих лицензий, так как распространяется как открытый программный продукт.

1С-Битрикс: Корпоративный портал

http://www.1c-bitrix.ru/products/intranet/

Портал включает более 500 готовых идей для интранет-ресурсов. В их числе:

- коллективная работа и социальные сети;
- автоматизация бизнес-процессов;
- защищенное перекрестное информационное пространство для взаимодействия с «внешним» миром;
- автоматизация и планирование офисных операций;
- внутрикорпоративные коммуникации;
- управление корпоративной информацией (ECM, Enterprise Content Management);
- корпоративный поиск;
- обучение и тестирование сотрудников;

Портал легко интегрируется в ИТ-инфраструктуру компании, обладая большим набором штатных интерфейсов к различным сервисам и службам: Active Directory, Microsoft Office, «1С», импорт/экспорт данных в различных форматах.

Коммерческая система.

Задача для самоконтроля

Загрузить из Интернет любую из вышеперечисленных систем, основанных на свободной лицензии и установить на свой компьютер, используя Denwer.

Заключение

В прочитанном вами пособии рассмотрены все основные технологии, используемые сегодня для построения веб-сайтов за исключением, возможно, технологий AJAX (которую несложно изучить самостоятельно, зная основы JavaScript и PHP), и Flash (становится неактуальной после выпуска стандарта HTML 5). Их обзор позволит самостоятельно создавать простейшие сайты, публиковать их в Интернет и корректно составлять технические задания на более масштабные системы.

Однако стоит помнить, что Интернет развивается семимильными шагами: выходят новые версии языков, дающие новые возможности; появляются новые скриптовые языки, упрощающие создание веб-приложений как для программистов, так и для рядовых администраторов доменов; выходят новые версии обозревателей Интернет, поддерживающие новые стандарты. Таким образом, необходимо обязательно учитывать самые последние тенденции в данной области.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Конкурсные работы «Доменная граматика». URL: http://www.nic.ru/news/2006/konkurs_part.html
2. *Лукач Ю.С.* Справочник Веб-разработчика. URL: http://wdh.suncloud.ru/
3. HTML-справочник: [сайт]. URL: http://html.manual.ru/
4. *Лебедев А.* .ру/Ковводство. URL: http://www.artlebedev.ru/kovodstvo /
5. *Сагалаев И.* Учебник. URL: http://softwaremaniacs.org/blog/category/primer/
6. Интернет-университет информационных технологий: [сайт]. URL: http://www.intuit.ru/
7. javascript.ru: [сайт]. URL: http://javascript.ru
8. CITForum: [сайт]. URL: http://www.citforum.ru
9. Видеолекции о применении phpMyAdmin. URL: http://www.webhosting.uk.com/phpmyadmin-tutorials.php

Приложение 1
Пример реализации статической HTML-страницы

П.1.1. Гипертекстовая часть (index.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml">

    <head>
        <title>Школа №111 - Главная страница</title>
        <meta http-equiv="Content-Type"
                content="text/html; charset=windows-1251" />
        <link rel="stylesheet" type="text/css" media="all"
                href="common/common.css" />
    </head>

    <body>

    <div id="header">
        <a    href="../index.html"><img    src="common/schoolimg.gif"
                alt="Школа №111" width="142" height="75"/></a>
        <div id="name">Школа №111</div>
    </div>

    <div id="primnav">
        <a href="about.html">Краткая справка</a> |
        <a href="history.html">История Школы</a> |
        <a href="life/">Жизнь Школы</a> |
        <a href="studies/">Учеба</a> |
        <a href="teachers/">Наши учителя</a> |
        <a href="school_leaver/">Выпускники</a> |
        <a href="photos/">Фотогалереи</a> |
        <a href="creative/">Творчество</a> |
        <a href="news/">Новости</a>
```

```
</div>


<div id="rightnav">
      <h1 style="text-align:center;">Контакты</h1>
   <p>123456, Россия, Волжск, ул.Ленина, д.10<br>
      Тел: (8212) 12-34-56<br>
      Факс: (8212) 12-34-56 <br>
      <a href="mailto:sch111@sch111.ru">
            sch111@sch111.ru</a><br>
      <a href="about.html#map">как нас найти</a></p>
</div>

<div  align="center"><img  src="img/school_main.jpg"  width="400"
       height="225" border="0" vspace="10" /></div>

<div id="footer">
      &copy; Школа №111, 2006
</div>
</body>
</html>
```

П.1.2. Каскадные таблицы стилей (common.css)

```
body {
      background: #FFF;
      font: 100%/1.3 Arial;
}


#header img {
      border: 2.5px solid #FFF;
      margin: 0 25px;
      float:left;
}


#header #name {
```

```
        margin: 15px 0;
        padding: 10px 0;
        text-align: left;
        font: 100%/1 Verdana;
        font-size: 30px;
        background: #66CC66;
        border-bottom: 1px solid #000;
        top:0;
        color: #FFF;
}


#primnav {
        border-bottom: 2px solid #CCC;
        font: 80%/1.3 Verdana;
        padding: .5em;
        text-align:center;
}

#primnav a {
        font-weight:bold;
}

#leftnav {
        border-right: 1px dashed #CCC;
        float: left;
        width: 150px;
        text-align:center;
        padding: 5px 10px;
        margin: 5px 1em 1em 0;
        background: #FFF;
}

#rightnav {
        border-left: 1px dashed #CCC;
        float: right;
        width: 170px;
        text-align:center;
```

```
        padding: 5px 10px;
        margin: 5px 0 1em 1em;
        background: #FFF;
}

#content {
        text-align:justify;
        text-indent: 2em;
        margin-left:190px;
}

#footer {
        border-top: 2px solid #CCC;
        clear: both;
        font: 80%/1.3 Verdana;
        text-align: center;
        padding: .5em 0;
}

/* headings */
h1, h2, h3, h4, h5, h6 {
        font-family: "Verdana";
        font-weight: bold;
        border-bottom: 1px dashed #CCC;
        text-indent:0;
        text-align:left;
}
h1 {
        font-size: 130%;
        margin: .6em 0;
}
h2 {
        font-size: 120%;
        margin: .5em 0;
}
h3 {
        font-size: 110%;
        margin: .4em 0;
```

```
}
h4 {
	font-size: 100%;
	margin: .3em 0;
}

/* tables */
table {
	background: #FFF;
	border-collapse: collapse;
	border: none;
	margin: 0 0 1em 0;
	table-layout: fixed;
	width: 100%;
	width:expression(document.body.offsetWidth-200);
}
thead, th {
	background: #EEE;
}
th,
td {
	border: 1px solid #CCC;
	padding: .1em .2em;
	text-align: left;
	text-indent:0;
}

/* links */
a:link {
	color: #00C;
	text-decoration : none;
}
a img:link {
	color: #000;
}
a:visited {
	color: #009;
	text-decoration : none;
```

```
}
a:hover {
	text-decoration : underline;
}
a:active {
	color: #F00;
}

small {
	color: #CCC;
	font-size:80%
}
```

П.1.3. Результат

ПРИЛОЖЕНИЕ 2
ПРИМЕРЫ ЧАСТО ИСПОЛЬЗУЕМЫХ СКРИПТОВ JAVASCRIPT

П.2.1. Проверка форм

Подключаемый скрипт test_form.js

```
function sendform () {
        // если массива с именами проверки нет, то форма
        // не отправляется
        ret_msg = elemListTest ? true : false;
        for (id in elemListTest)
                if (!document.getElementById(id).value) {
                // показать сообщение
                        show_msg(id);
                        ret_msg = false;
                } else if (document.getElementById(id + "-msg")) {
                // скрыть сообщение
        document.getElementById(id + "-msg").style.display = "none";
                }
        return ret_msg;
}

function show_msg(id) {
        // сначала проверка – создано ли уже сообщение
        if (document.getElementById(id + "-msg")) {
        //если создано - показать
        document.getElementById(id + "-msg").style.display = "";
        } else { // создать сообщение
                var br = document.createElement('br');
                var err_msg = document.createElement('span');
                err_msg.setAttribute("id", id + "-msg");
                err_msg.setAttribute("style",
                "background-color:#FF9999; border: red 1px solid;
                padding:2px 6px; margin:5px 0;");
err_msg.appendChild(document.createTextNode(elemListTest[id]));
```

```
document.getElementById(id).parentNode.appendChild(br);
document.getElementById(id).parentNode.appendChild(err_msg);
        }
}
```

Вызов функции в HTML-документе для любой формы

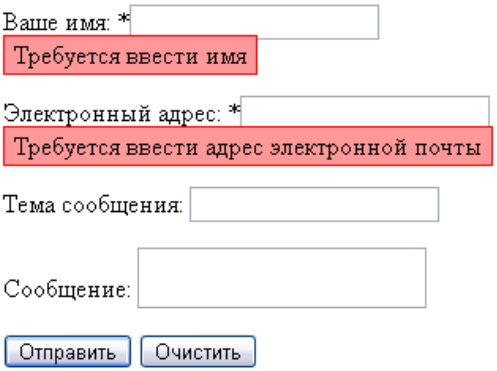
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head>
        <meta http-equiv="Content-Type" content="text/html;
                charset=windows-1251">
        <title>Проверка формы</title>
        <script type="text/javascript" src="test_form.js"></script>
</head><body>

<form action="#" onSubmit="return sendform();">
        <p>Ваше имя: *
                <input type="text" name="name" id="name"></p>
        <p>Электронный адрес: *
                <input type="text" name="email" id="email"></p>
        <p>Тема сообщения: <input type="text" name="subject"></p>
        <p>Сообщение: <textarea name="message"></textarea></p>
        <input type="submit" value="Отправить">
        <input type="reset" value="Очистить">

<script type="text/javascript">
        elemListTest = { "name": "Требуется ввести имя", "email":
                "Требуется ввести адрес электронной почты" }
</script>

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

Вид после выявления ошибок



П.2.2. Меню-телепортатор

```
<select onChange="top.location.href =
this.options[this.selectedIndex].value">
    <option value="#">- выберите поисковую машину -</option>
    <option value="http://yandex.ru"> Yandex</option>
    <option value="http://google.ru">Google</option>
</select>
```

П.2.3. Смена изображения при наведении мыши

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head>
        <meta http-equiv="Content-Type"
                content="text/html; charset=windows-1251">
        <title>Смена изображений при наведении мыши</title>

<script type="text/javascript">

var revert = new Array(); // хранилище второй ссылки
// хранилище ссылок на общую часть названия изображений
var inames = new Array('q', 'w');
```

```
// Предварительная загрузка изображений
// (пользователь увидит страницу
// только тогда, когда все изображения будут загружены)

if (document.images) {
        var flipped = new Array();
        for(i=0; i< inames.length; i++) {
                flipped[i] = new Image();
                flipped[i].src = inames[i]+"2.gif";
        }
}

function over(num) {
        if(document.images) {
                revert[num] = document.images[inames[num]].src;
                document.images[inames[num]].src = flipped[num].src;
        }
}

function out(num) {
        if(document.images) document.images[inames[num]].src =
                revert[num];
}

</head><body>

<img src="media/q1.gif" name="smile" onMouseOver="over(0)"
                                        onMouseOut="out(0)">

<br>

<img src="media/w1.gif" name="smile" onMouseOver="over(1)"
                                        onMouseOut="out(1)">

</body></html>
```