Ю.Ю. ГРОМОВ, Н.А. ЗЕМСКОЙ,
О.Г. ИВАНОВА, А.В. ЛАГУТИН

ОСНОВЫ WEB-ИНЖИНИРИНГА

Часть 1

РАЗРАБОТКА КЛИЕНТСКИХ ПРИЛОЖЕНИЙ

ИЗДАТЕЛЬСТВО ТГТУ

УДК 004.42(075)
ББК ←973-018.4я73
О753

Р е ц е н з е н т ы :

Доктор физико-математических наук, профессор

Е.Ф. КУСТОВ

Доктор физико-математических наук, профессор
В.Ф. Крапивин

О753 Основы Web-инжиниринга. В 2 ч. Ч. 1 : Разработка клиентских приложений : учебное пособие / Ю.Ю. Громов, Н.А. Земской, О.Г. Иванова, А.В. Лагутин. – Тамбов : Изд-во Тамб. гос. техн. ун-та, 2007. – 240 с. – 100 экз. – ISBN 978-5-8265-0645-5.

Даны основы проектирования клиентской части Web-приложений, в частности, основы Web-проектирования, создания HTML-документов, технологии каскадных таблиц стилей, JavaScript.

Утверждено Ученым советом университета для студентов высших учебных заведений, обучающихся по специальностям 230201 – Информационные системы и технологии, 090105 – Комплексное обеспечение информационной безопасности автоматизированных систем и 230105 – Программное обеспечение вычислительной техники и автоматизированных систем.

УДК 004.42(075)
ББК ←973-018.4я73

ISBN 978-5-8265-0645-5

Министерство образования и науки Российской Федерации

ГОУ ВПО "Тамбовский государственный технический университет"

Ю.Ю. ГРОМОВ, Н.А. ЗЕМСКОЙ,
О.Г. ИВАНОВА, А.В. ЛАГУТИН

ОСНОВЫ WEB-ИНЖИНИРИНГА

Часть 1

Разработка клиентских приложений

Допущено Учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности 230201 «Информационные системы и технологии»

Тамбов
♦ Издательство ТГТУ ♦
2007

Учебное издание

Громов ЮРИЙ ЮРЬЕВИЧ,

Земской НИКОЛАЙ АЛЕКСАНДРОВИЧ,

Иванова ОЛЬГА ГЕННАДЬЕВНА,

Лагутин АНДРЕЙ ВЛАДИМИРОВИЧ

ОСНОВЫ WEB-ИНЖИНИРИНГА

Часть 1

разработка клиентских приложений

Учебное пособие

Редактор Т.М. Г л и н к и н а
Инженер по компьютерному макетированию М.А. Ф и л а т о в а
Корректор О.М. Я р ц е в а

Подписано к печати 16.11.2007
Формат 60 × 84 / 16. 13,95 усл. печ. л. Тираж 100 экз. Заказ № 735

Издательско-полиграфический центр
Тамбовского государственного технического университета,
392000, Тамбов, Советская, 106, к. 14

ВВЕДЕНИЕ

Быстрое развитие сети Интернет привело к появлению массы литературы об Интернет-технологиях. Следует заметить, что авторы зачастую рассматривают материал на уровне описания меню и основных возможностей отдельно взятой программы либо ограничиваются формальным описанием одной из существующих технологий, как правило – языка HTML. Иногда практически весь материал излагается методом "step-by-step" (шаг за шагом), что тяжело воспринимается читателями.

В учебном пособии излагаются основы создания web-публикаций с помощью технологии "клиент-сервер" и их последующего размещения и сопровождения на web-серверах. Главной задачей предлагаемого пособия является знакомство читателей с практическими вопросами создания как клиентских приложений web-публикаций, так и серверных программ.

В пособии представлен весь спектр технологий создания различных Web-документов, начиная от простейших статических документов, использующих "чистый" HTML-код, до сложных документов, использующих динамическую генерацию содержимого, средства интерактивного общения с пользователем, базы данных, мультимедиа-объекты и др.

Учебное пособие состоит из двух частей. В первой части пособия рассматриваются основные компоненты web-дизайна, такие как язык гипертекстовой разметки документов – HTML, каскадные таблицы стилей – CSS, язык JavaScript, используемый при разработке сценариев, выполняемых при создании интерактивных элементов web-страниц. Излагается технология применения перечисленных компонентов при построении web-сайтов. При этом затрагиваются вопросы формирования концепции развития сайта, организации его структуры и системы навигации, а также размещения в сети Интернет.

Технологии создания, модификации и публикации электронных документов в Интернет рассмотрены в пособии на большом количестве примеров, что позволяет использовать его для самостоятельного изучения курса или выполнение практических упражнений в "исследовательском" режиме под руководством преподавателя. Все разделы сопровождаются вопросами контроля полученных знаний и упражнениями.

Пособие предназначено для студентов вузов, обучающихся по специальностям: 230201 – Информационные системы и технологии, 090105 – Комплексное обеспечение информационной безопасности автоматизированных систем, 230105 – Программное обеспечение вычислительной техники и автоматизированных систем и может быть использовано при изучении таких дисциплин, как "Информационные технологии", "Проектирование информационных систем" а также дисциплин специализации, связанных с использованием Интернет-технологий.

1. ОСНОВЫ WEB-ПРОЕКТИРОВАНИЯ

Информационное проектирование (information design) – это процесс организации содержания и представления его в форме, наиболее удобной для восприятия целевой аудиторией узла.

Для успешного воплощения первоначального замысла необходимо решить следующие задачи:

1. Определение общей концепции и предназначения публикации.
2. Определение категорий потенциальных посетителей сайта.
3. Выбор общего стиля (не только визуального) публикаций.
4. Разработка структуры публикаций (с учетом внешних и внутренних ссылок, а также возможной последующей модификации сайта).
5. Разработка главной страницы.
6. Разработка страниц, на которые существуют (или могут существовать) ссылки извне.
7. Разработка основных страниц.
8. Размещение публикации в Сети и регистрация в поисковых системах.
9. Анализ рейтинга и принятие решения о внесении изменений в проект.
10. Пересмотр ранее принятых решений, относящихся к одному (или ко всем) из пунктов с первого по седьмой.

1.1. ОПРЕДЕЛЕНИЕ ОБЩЕЙ КОНЦЕПЦИИ И ПРЕДНАЗНАЧЕНИЯ ПУБЛИКАЦИИ

Выбор общей концепции и предназначения публикации влияет на архитектуру, информационное наполнение и стилевое оформление сайта.

Поэтому процесс разработки должен начинаться с попытки классифицировать будущий проект. Учитывая разнообразные и постоянно меняющиеся формы представления Web-материалов, задача классификации является достаточно сложной. Тем не менее, все виды Web-публикаций можно упорядочить по трем основным признакам: объему, целевому предназначению и используемым технологиям.

Объем публикации может меняться от одной страницы до нескольких тысяч и, по большому счету, не является показателем ни качества, ни популярности публикации. Тем не менее, публикация, содержащая достаточно большое количество страниц (сколько именно, не знает никто), переходит на качественно новый уровень и становится сайтом.

В последнее время появился еще один термин – портал, под которым в общем случае понимается объединение нескольких тематических направлений. В связи с этим объем портала должен существенно превосходить объем сайта. Существует и более развернутая трактовка этого термина: *портал* – это сайт, предоставляющий посетителю персонализированную начальную страницу, бесплатные услуги электронной почты, новостной и развлекательный сервисы, а также поисковый механизм.

Несмотря на указанные различия в этих понятиях, все они представляют собой совокупность связанных между собой гипертекстовых документов и поэтому часто в качестве обобщенного термина для обозначения любой Web-публикации, содержащей более одной страницы, используют слово "сайт".

Создание Web-материалов может преследовать одну из следующих *целей*:

– общение;

– информирование и развлечение;

– обучение и консультации;

– электронная коммерция.

Большинство страниц, размещенных в настоящее время в Интернете, относится к первой категории. Это так называемые "домашние странички". Их содержание может быть самым разнообразным и отражает как цель их создания, так и личные качества автора.

Под информационными сайтами будем понимать публикации, адресованные достаточно широкой аудитории и содержащие актуальную информацию по соответствующей тематике.

Форма подачи материала на информационных сайтах зависит от предметной области и характеристик той аудитории, которой адресована информация. Владельцами информационных сайтов чаще всего являются различные организации коммерческого и некоммерческого характера и общества. Основное содержание коммерческих сайтов составляет рекламная информация, научные организации и учебные заведения публикуют аналитические материалы, свои достижения и результаты. Особое место среди информационных сайтов занимают поисковые системы.

В настоящее время Интернет предоставляет очень широкие возможности для реализации учебных курсов в системах удаленного обучения и консультирования.

Все более популярным применением Интернет-технологий становится электронный бизнес. В настоящее время можно выделить три основных вида электронной коммерции:

– торговля и услуги, ориентированные на частного потребителя;

– фондовые торги и инвестиционные сделки, в которых с обеих сторон участвуют юридические лица;

– Интернет-банкинг – предоставление банковских услуг через Интернет.

Ограничимся комментариями только по первому направлению. Эта форма бизнеса реализуется через так называемые электронные магазины.

Стандартный электронный магазин поддерживает следующие функции:

– регистрация посетителя;

– знакомство с ассортиментом предлагаемых товаров и услуг;

– предоставление более подробной информации по заинтересовавшим посетителя товарам;

– регистрация заказа (наполнение корзины посетителя).

Основной проблемой при реализации полноценного электронного магазина на сегодняшний день является осуществление безналичных электронных платежей за выбранные товары. Поэтому большая часть магазинов, представленных в российском Интернете, правильнее было бы называть "электронными витринами".

Считается, что Web-узлы прошли в своем развитии три стадии, напрямую связанные с эволюцией *технологии* их создания.

Для Web-узлов первого поколения характерна однотипная структура – обычный сайт представлял собой последовательность текста и картинок. При этом единственной формой интерактивного взаимодействия читателя с документом являлись гиперссылки.

Страницы узлов второго поколения уже содержали интерактивные элементы, обеспечивающие более активное участие пользователя в формировании облика документа. Такие изменения стали возможны благодаря расширениям HTML и его совместному использованию с клиент-серверными технологиями. Причем основная работа по обработке действий пользователя выполнялась на сервере, что обуславливало достаточно высокую сложность программ сценариев и низкую скорость взаимодействия читателя с содержимым страниц. Основными инструментами для создания узлов и первого, и второго поколения являлись текстовые редакторы.

Последующее развитие технологии создания Web-материалов шло по трем основным направлениям:

1) разработка инструментов, которые позволили бы свести к минимуму ручное кодирование и одновременно были бы пригодны для выполнения операций по сопровождению Web-публикаций;

2) перераспределение функций по обеспечению интерактивности страниц между сервером и клиентом в сторону последнего;

3) расширение диапазона мультимедийных компонентов, включаемых в состав публикаций, и упрощение такой интеграции.

1.2. ОПРЕДЕЛЕНИЕ КАТЕГОРИЙ ПОТЕНЦИАЛЬНЫХ ПОСЕТИТЕЛЕЙ САЙТА

Существует несколько вариантов классификации пользователей интерактивных систем. Рассмотрим наиболее общий вариант.

1. **Человеческие потребности** обозначают, в частности, потребность быть понятым партнером по диалогу; например, пользователь хочет иметь возможность личного развития или хочет изменить окружение и при этом не хочет оказаться ограниченным в своем поведении.

2. **Навыки пользователя** состоят из моторных навыков, лингвистических навыков, навыков в общении и навыков в решении задач.

3. **Свойства личности** – это, например, творческие способности, подверженность ошибкам, способность к обучению, терпеливость, устойчивость к стрессу и т.д.

4. **Уровень компьютерной грамотности**: здесь обычно различают программирующих и непрограммирующих пользователей, а среди последних, в свою очередь, выделяют три категории:

– подготовленные пользователи, решающие творческие задачи, – аналитики и исследователи (т.е. пользователи, последовательность действий которых сложно формализовать);

– подготовленные пользователи, выполняющие рутинные операции (т.е. пользователи, последовательность действий которых является достаточно устойчивой);

– случайные (или "наивные") пользователи, обладающие минимальным уровнем компьютерной грамотности.

5. **Подготовка в прикладной области** пользователя влияет на использование языка (например, профессиональной терминологии) и применяемые методы решения задач.

6. **Причина** пользования системой может быть:

– обязательной как неотъемлемая часть работы;

– необязательной как дополнительная составляющая профессиональной деятельности;

– обязательной с точки зрения личных потребностей (например, когда определенную информацию можно получить только с помощью компьютера);

– необязательной в частной жизни (например, в качестве развлечения).

7. **Отношение к системе и ожидания от работы с ней** определяются уровнем компьютерной грамотности и причиной ее использования; оно может быть нейтральным, положительным и негативным.

8. **Целями пользователя** могут быть: решить определенную задачу с помощью компьютера или научиться работать с системой.

9. **Ограничения во времени**: независимо от характеристик системы пользователь может быть вынужден приспосабливаться к ограничениям во времени, исходящим от задачи или контекста работы.

1.3. ВЫБОР ОБЩЕГО СТИЛЯ ПУБЛИКАЦИИ

Стиль публикации определяется не только визуальным оформлением страниц сайта, но также стилем изложения его содержания (контента – content) и средствами взаимодействия посетителя с этим содержанием. Эти три составляющие формируют пользовательский интерфейс публикации.

Какими же свойствами должен обладать "хороший" интерфейс?

1. Естественность. Естественный интерфейс не вынуждает пользователя существенно изменять привычные для него способы решения задачи. Применительно к Web-интерфейсу это означает, в частности, что представленная на странице информация не должна требовать пояснений. Целесообразно также сохранить систему обозначений и терминологию, используемые в данной предметной области. Кроме того, какие бы элементы не содержала страница, пользователь должен однозначно идентифицировать их предназначение.

2. Согласованность интерфейса. Она позволяет пользователям применять знания и навыки, полученные ими при работе с локальными приложениями или с другими Web-ресурсами, при посещении данного сайта, и благодаря этому фокусировать внимание на решаемой задаче, а не тратить время на уяснение различий в использовании тех или иных элементов управления, способах навигации и т.д. Кроме того, согласованность интерфейса должна проявляться в оформлении страниц сайта. Речь идет о том, что однотипные элементы на всех страницах должны выглядеть одинаково и размещаться в одной и той же позиции.

3. Дружественность интерфейса. Пользователи обычно изучают особенности работы с новым программным продуктом методом проб и ошибок. Попав на новый сайт, они также будут использовать этот подход. Следовательно, на каждом этапе работы интерфейс должен разрешать только соответствующий набор действий и предупреждать пользователей о последствиях тех действий, которые могут оказаться для них неожиданными или неблагоприятными, еще лучше, если у пользователя существует возможность отменить или исправить выполненные действия. Ни в коем случае нельзя умышленно обманывать посетителя, заманивая его с помощью различных ухищрений туда, куда он совсем не хочет (например, на сайты коммерческих организаций).

4. Принцип обратной связи. Каждое действие пользователя, посетившего ваш сайт, должно получать визуальное, а иногда и звуковое подтверждение того, что оно (действие) воспринято системой; при этом вид реакции должен учитывать природу выполненного действия. Например, общепринятым правилом является изменение цвета использованных гиперссылок.

5. Простота интерфейса. Один из возможных путей обеспечения простоты – представление на экране информации, минимально необходимой для определения пользователем дальнейшей последовательности действий или для усвоения того, что ему предлагается. Применение гипертекстовой организации данных в полной мере позволяет реализовать такой подход. Практика показывает, что в большинстве случаев информация сайта может быть скомпонована таким образом, чтобы очередная ее порция умещалась на одном экране. В наибольшей степени это пожелание относится к начальной странице сайта.

Другой путь к созданию простого, но эффективного интерфейса – размещение и представление элементов на экране с учетом их смыслового значения и логической взаимосвязи. Здесь важную роль играет как пространственная компоновка элементов (группирование, выравнивание), так и соответствующий набор визуальных атрибутов: наиболее важные элементы должны выделяться на общем фоне. Такой подход позволяет использовать в процессе работы ассоциативное мышление пользователя.

6. Гибкость интерфейса. Возможность настройки пользовательского интерфейса является отличительной чертой хорошей программы. На Web-страницах элементы настройки интерфейса, хотя и нечасто, но встречаются. Наиболее популярным вариантом настройки является возможность выбора языка и способа кодировки символов для отображения текстового содержимого страниц. Реже предлагается выбрать размер и способ вывода графических изображений. И совсем редко предла-

гается выбрать один из нескольких вариантов общего дизайна страницы (в простейшем случае – это выбор между текстовым и графическим представлением материалов).

Существует несколько веских причин предоставления пользователям права "собственноручной" настройки страниц:

– необходимость учета характеристик аппаратуры и соединения с Интернетом;

– необходимость учета особенностей применяемого пользователем программного обеспечения;

– языковые предпочтения и индивидуальные психофизиологические характеристики людей, для которых эти страницы созданы.

При размещении элементов настройки интерфейса следует использовать следующие рекомендации:

– о возможности настройки страницы пользователь должен быть уведомлен сразу, как только он ее открывает;

– если страница содержит много графики (которая загружается значительно медленнее текста и поэтому появляется в последнюю очередь), следует предусмотреть текстовый вариант переключателя, реализуемого в форме меню или списка. Таким образом оформляются, например, переключатели используемого языка и кодировки символов текста страницы. В некоторых случаях переключатели языка имеют вид небольших изображений с символикой соответствующих национальностей (флаг государства). Этот вариант хорош своей наглядностью и компактностью, но предполагает определенный уровень образованности и сообразительности пользователя. Если в браузере отключен вывод графики, предлагаемый вариант должен предусматривать вывод альтернативного текста.

1.4. РАЗРАБОТКА СТРУКТУРЫ ПУБЛИКАЦИИ

По итогам выполнения первых трех этапов может быть определена структура узла, которая имеет два представления:

– в виде файловой структуры;

– в виде навигационной схемы.

При разработке структуры публикации целесообразно использовать подход "проектирование сверху вниз". Применительно к Web-публикации его суть заключается в том, что сначала определяется состав публикации на уровне крупных разделов, содержание которых постепенно детализируется и уточняется. В такой же последовательности устанавливаются и связи между разделами публикации.

Хотя обе формы представления структуры узла удобнее разрабатывать параллельно, поочередно уточняя и корректируя их, начинать все же лучше с определения файловой структуры проекта.

При выборе файловой структуры проекта следует придерживаться определенных правил:

– взаимосвязанные страницы следует размещать в одной папке;

– для упорядочения информации внутри больших разделов следует использовать вложенные папки;

– файлы разных типов следует размещать в отдельных папках; например, графические файлы – в папке Images, звуковые файлы – в папке Sound и т.п.; если таких файлов много, их нужно распределить по вложенным папкам, имена которых должны соответствовать наименованиям страниц, где эти файлы используются;

– одна и та же структура должна быть использована для проекта, размещенного локально, и для его копии, развернутой на удаленном сервере.

При выборе оптимального размера страницы определяющим является условие, чтобы каждая страница была завершена с логической точки зрения. Поэтому не следует делить изложение некоторой идеи на несколько страниц, чтобы сократить размер каждой из них. Однако при выборе размера документа следует учитывать следующие факторы:

– длинный документ требует больше времени для пересылки и отображения браузером;

– читателю сложнее работать с большим по объему документом: он не может быстро переходить к нужной его части и возвращаться назад, используя полосу прокрутки, в то время как переход по ссылкам может оказаться значительно эффективнее;

– чем длиннее документ, тем менее заметны в нем слова, заданные в запросе, и следовательно, страница будет ниже в результатах поиска при прочих равных условиях; более того, роботы некоторых поисковых систем просто не обрабатывают слишком длинные документы, например, максимальный размер документов для роботов Rambler составляет 200 кбайт.

В то же время чрезмерное увлечение созданием ссылок также может помешать как создателю сайта, так и посетителям.

Продуманная навигационная схема сайта должна предоставлять посетителям важную возможность – оценить, какая информация находится на узле и насколько быстро можно до нее добраться. Находясь на любой странице узла, посетители должны представлять свое местоположение, знать, что они могут сделать на этой странице и куда пойти дальше.

Для выполнения этого условия достаточно соблюдать несколько правил:

– на главной странице сайта обязательно должна быть представлена карта сайта, причем не только в графической, но и в текстовой форме;

– на всех последующих страницах должны присутствовать ссылки на главную страницу и/или на первую страницу данного раздела (на которой, в свою очередь, должно быть представлено содержание раздела);

– при использовании рисунков в качестве ссылок они должны быть однотипными на всех страницах и располагаться в одной и той же позиции;

– рисунки, используемые в качестве ссылок, должны быть интуитивно различимы для читателей относительно других элементов страницы.

Хорошо спланированная структура является фундаментом для дальнейшего развития узла. Возможность наращивания информационного наполнения сайта без кардинального изменения его структуры называется **масштабируемостью** сайта. Масштабируемость – важное свойство любого сайта: сайт обязательно должен развиваться. Кроме того, масштабируемость узла является одним из условий согласованности его интерфейса: после того как посетители уже изучили структуру и схему навигации узла, они будут опираться на это знание во время последующих посещений.

"Законы разработчика":

– **Закон ссылок**: чем большее число ссылок на странице конкурирует за внимание пользователя, тем меньше вероятность, что он воспользуется хотя бы одной из них;

– **Закон щелчков**: чем больше щелчков мышью нужно сделать, чтобы добраться до страницы, тем меньше на ней окажется посетителей.

1.5. ПРОЕКТИРОВАНИЕ ГЛАВНОЙ СТРАНИЦЫ

Главная страница должна дать посетителю представление об общей направленности сайта, а также о его структуре и способах навигации.

При выборе способа визуальной компоновки и объема информации на странице следует помнить психофизиологические особенности посетителей. Например о том, что человек не способен идентифицировать одновременно более пяти – семи видимых предметов, что графическая информация воспринимается легче текстовой и т.д. Вместе с тем, чрезмерно насыщенная графикой страница будет обрабатываться браузером значительно дольше.

Лучше, если все содержимое главной страницы уместится на одном экране, поскольку при перелистывании информации целостность ее восприятия в значительной степени нарушается. При этом основные элементы навигации по узлу целесообразно размещать в верхней части страницы. Объясняется это тем, что элементы, расположенные внизу, часто воспринимаются пользователями как второстепенные. А если они окажутся за пределами первого экрана, то посетитель вообще их может не заметить.

Тот элемент страницы, с которого, по мнению разработчика, посетитель должен начать знакомство с сайтом, следует разместить либо в ее центре, либо в верхнем левом углу. Если выбран второй вариант, то геометрические размеры этого элемента должны быть несколько больше, чем при его размещении в центре (вместо этого могут использоваться и другие средства привлечения внимания – повышенная яркость, анимация и т.п.).

По сложившейся традиции Web-строительства оглавление или навигационная схема сайта помещаются в левой части страницы.

1.6. ТЕСТИРОВАНИЕ ПРОЕКТА, РАЗВЕРТЫВАНИЕ НА СЕРВЕРЕ И СОПРОВОЖДЕНИЕ

Разработка каждой очередной страницы публикации должна завершаться ее тестированием, которое предполагает выполнение следующих действий:

– проверку орфографии и синтаксиса текстового содержимого страницы;

– проверку корректности HTML-кода;

– проверку корректности и "дееспособности" всех ссылок, используемых на странице;

– оценку времени загрузки страницы.

Размещение сайта на Web-сервере заключается в копировании файлов и папок проекта на жесткий диск компьютера, который будет использоваться в качестве Web-сервера. Трудности, которые могут возникнуть при выполнении этой процедуры, обусловлены, как правило, отличием файловой организации Web-сервера от файловой организации того компьютера, на котором создавался сайт.

С момента размещения сайта начинается этап его сопровождения. Сопровождение сайта предполагает выполнение нескольких важных процедур:

– обновление страниц публикации с целью улучшения их внешнего оформления или содержания:

– добавление, перемещение или удаление страниц;

– поддержание корректности ссылок;

– загрузка новых или измененных файлов на Web-сервер, а также удаление существующих.

В последнее время все большее распространение получил еще один термин, связанный с обеспечением и поддержанием работоспособности узлов – Web-хостинг. Это понятие является более широким, чем сопровождение.

Web-хостинг предусматривает предоставление и обслуживание аппаратного обеспечения, приложений, поддержку целостности информационного наполнения, организацию защиты и установление высокоскоростных соединений с Internet для Web-узла.

Как правило, Web-хостингом занимаются либо компании-провайдеры, либо организации, поддерживающие свой собственный узел.

Правильный выбор провайдера, предоставляющего доступ к Web-странице, позволит с максимальным удобством получать необходимую информацию. Кроме того, поддержка Web-сервером специальных возможностей значительно облегчит разработку Web-узла.

На что следует обратить внимание при выборе провайдера, размещающего ваш Web-узел на своем сервере?

Пропускная способность каналов. Чтобы посетителям не пришлось слишком долго ждать загрузки страниц, провайдер должен обладать надежным высокоскоростным соединением порядка 1–2 Мбит в секунду.

Поддержка сервером провайдера SSI (Server Side Includes, вставки на стороне сервера). Использование SSI позволяет Web-серверу вставлять небольшие объемы динамических данных непосредственно в пересылаемый пользователю HTML-документ. Запрошенная HTML-страница "просматривается" в поисках элементов SSI. Обнаружив такой элемент, сервер вставляет требуемую динамическую информацию. С помощью SSI можно включать один файл в состав другого, исполнять CGI-сценарии и передавать другую информацию. Необходимо уточнить, какие именно функции SSI поддерживаются на сервере провайдера.

Поддержка сервером провайдера CGI-сценариев. CGI (Common Gateway Interface, общий шлюзовой интерфейс) – спецификация, позволяющая Web-серверу выполнять произвольные прикладные программы. В результате работы таких

программ (сценариев, или "скриптов") создаются HTML-документы. С помощью CGI-сценариев могут приниматься данные от пользователя, они позволяют организовать диалог на Web-страницах, запросы к базам данных и т.д. Создать CGI-сценарий можно с помощью любого популярного языка программирования: Perl, Basic, C, C++, Pascal и т.п.

Поддержка моментальной перекодировки. К сожалению, для русского языка в Internet при работе на разных платформах (Windows, Mac, Unix и т.д.) приняты различные кодировки. Чтобы пользователю было легко просматривать страницы, Web-сервер провайдера должен уметь автоматически перекодировать документы в зависимости от поступившего запроса. В противном случае либо содержание Web-узла для некоторых посетителей будет нечитаемым, либо придется обеспечивать несколько копий Web-узла – по одной на каждую поддерживаемую кодировку.

Способ обновления страниц. Обычно страницы обновляются по протоколу FTP (File Transfer Protocol, протокол передачи файлов). Некоторые FTP-клиенты позволяют работать с файлами на компьютере провайдера так же, как с собственным диском, – копировать, удалять, переименовывать и т.п.

Как правило, возможность размещения Web-узла провайдер предоставляет своим пользователям за небольшую плату или бесплатно.

Существуют службы, которые предоставляют место под Web-узлы бесплатно вместе с адресом электронной почты и другими услугами. Как правило, условием такого "бесплатного" размещения является выделение на страницах некоторого места под рекламу. Кроме того, накладываются ограничения на размер файлов.

Контрольные вопросы

1. Сформулируйте задачи, которые необходимо решить при информационном проектировании.
2. По каким параметрам можно классифицировать Web-публикации?
3. Дайте определение таким понятиям, как страница, сайт, портал.
4. Сформулируйте основные цели создания Web-материалов.
5. Перечислите свойства эффективного интерфейса Web-публикации.
6. Каковы особенности проектирования структуры публикации?
7. Каким образом можно представить структуру публикации?
8. Какие правила следует соблюдать при разработке файловой структуры публикации?
9. Назовите особенность проектирования главной страницы.
11. Определите, к какому виду будет относиться Ваша публикация, для какого пользователя она рассчитана.
12. Разработайте навигационную и файловую структуры Вашей публикации.

2. СОЗДАНИЕ ДОКУМЕНТОВ HTML

2.1. СИНТАКСИС И СТРУКТУРА HTML

Одним из компонентов технологии создания распределенной гипертекстовой системы World Wide Web стал язык гипертекстовой разметки HTML (Hypertext Markup Language, язык разметки гипертекста), разработанный Тимом Бернерсом-Ли на основе стандарта языка разметки печатных документов – SGML (Standard Generalised Markup Language, стандартный обобщенный язык разметки). Дэниел В. Конноли написал для него Document Type Definition – формальное описание синтаксиса HTML в терминах SGML.

Язык HTML позволяет размечать электронный документ, который отображается на экране с полиграфическим уровнем оформления; результирующий документ может содержать самые разнообразные метки, иллюстрации, аудио- и видеофрагменты и так далее. В состав языка вошли развитые средства для создания различных уровней заголовков, шрифтовых выделений, различные списки, таблицы и многое другое. В качестве основы был выбран обычный текстовый файл.

Таким образом, гипертекстовая база данных в концепции WWW – это набор текстовых файлов, размеченных на языке HTML, который определяет форму представления информации (разметка) и структуру связей между этими файлами и другими информационными ресурсами (гипертекстовые ссылки). Гипертекстовые ссылки, устанавливающие связи между текстовыми документами, постепенно стали объединять самые различные информационные ресурсы, в том числе звук и видео; в результате возникло новое понятие – гипермедиа.

Такой подход предполагает наличие еще одного компонента технологии – интерпретатора языка. В World Wide Web функции интерпретатора разделены между Web-сервером гипертекстовой базы данных и интерфейсом пользователя. Сервер, кроме доступа к документам и обработки гипертекстовых ссылок, обеспечивает предпроцессорную обработку документов, в то время как интерфейс пользователя осуществляет интерпретацию конструкций языка, связанных с представлением информации.

Главной несущей конструкцией всех команд HTML является *тег* (который иногда называется *дескриптором* или *элементом).* Тег (от английского слова tag – ярлык, признак) – это элемент HTML, определяющий некоторое действие. Мощь тегов проявляется благодаря уточнениям, называемым *атрибутами* (известными также как *аргументы*), которые, в свою очередь, имеют *значения*.

Общая схема построения тега в формате HTML может быть записана в следующем виде:

"тег"=

<"имя тега" "список атрибутов">

содержание
</"имя тега">

Теги HTML подчиняются как частным, так и общим правилам. Первое правило гласит, что все стандартные теги находятся между символами "меньше" (<) и "больше" (>), например:

<HTML>

Между символами < > и тегом нет пробелов, как и между буквами, составляющими сам тег. Единственным исключением из этого правила являются теги комментария с более сложными ограничителями (<!-- и -->).

Следующее правило не предполагает такого последовательного выполнения. Оно утверждает, что теги имеют открывающие и закрывающие компоненты. В закрывающем компоненте применяются те же самые символы "меньше" и "больше", но перед тегом, чтобы показать, что он закрыт, добавляется прямая косая черта (/):

</HTML>

Это означает, что большинство стандартных тегов, после того как вы их открыли, в определенном месте кода следует закрывать. Это правило является не совсем точным, так как многие теги ему не подчиняются. Некоторые из них *могут* быть одиночными, например тег <P>, применяемый для абзацев. Его можно законно использовать в двух форматах: одиночном (<P>) и открывающем/закрывающем (<P> . . . </P>). Другие теги *не могут* использоваться с тегом закрытия, например <IMG>, для которого никогда не ставится закрывающий компонент.

К общим правилам относится и то, что для тегов необходимо выбрать один реестр символов и придерживаться его. Для единообразия в данном пособии большинство тегов записывается прописными буквами.

Другой важный момент, на который стоит обратить внимание, – это пробелы. Иногда они обязательно должны быть в некоторых местах "предложения" HTML, а иногда лишний пробел может так запутать браузер, что тот не сумеет показать никакой информации.

Атрибуты, как уже отмечалось, определяют действие тега. Большинство тегов могут прекрасно работать в одиночку, но есть и много таких, которые для выполнения своих функций *обязательно* должны иметь атрибуты.

Тег <HTML> и его закрывающий тег </HTML> показывают начало и конец HTML-документа. У этого тега нет атрибутов. Тег <BODY> со своим закрывающим тегом </BODY> определяют область кода, которая будет выведена в браузере. Хотя теоретически этот тег может работать в одиночку, к нему могут быть применены атрибуты, такие, как bgcolor для задания цвета фона или text для указания цвета текста.

Атрибуты должны находиться только в открывающем теге. Например, синтаксически верной является строка <BODY bgcolor="#FFFFFF">, которая закрывается с помощью тега </BODY>.

Зачастую атрибуты представляют собой обычные слова, а иногда – их сокращения. Среди атрибутов, являющихся целыми словами, следует отметить такие, как align (выравнивание), color (цвет), link (ссылка) и face (название шрифта). А среди примеров сокращений слов – src для "source" (источник) и vlink для "visited link" (посещенная ссылка).

Атрибуты следуют после тега и пробела с указанием значений атрибутов:

<BODY bgcolor...

В теге может быть несколько атрибутов. В таком случае синтаксис следует той же концепции: вначале тег, пробел, а затем атрибут. Атрибут получает значение, а затем перед следующим атрибутом снова ставится пробел:

<BODY bgcolor="#FFFFFF" text="#000000">

И так далее, до тех пор, пока не будут введены все атрибуты с соответствующими значениями.

Значения являются неотъемлемой частью атрибутов и в конечном итоге определяют сам тег. Задавая качественные или количественные показатели, значения уточняют способ выполнения некоторого действия.

Значения, как и атрибуты, могут представлять собой целое слово. Например, если используется тег раздела <DIV> и в этом разделе требуется выровнять информацию, то можно выбрать одно из нескольких значений, которые будут определять атрибут align. Этими значениями являются left (по левому краю), right (по правому краю), center (по центру) и justify (по ширине).

В результате может получиться такая строка: <DIV align="right">

Теперь вся информация в этом разделе будет выровнена по правому краю, так как сперва был использован тег <DIV>, указывающий начало раздела, затем был определен атрибут align со значением right. Это определение будет в силе до тех пор, пока раздел не будет закрыт с помощью тега </DIV>.

Некоторые значения являются числовыми, указывающими, например, количество пикселей или процентов, или размеры, определяемые браузерами; кроме того, они могут быть представлены, например, шестнадцатеричными числами, задающими цвет HTML. Пиксельное значение можно рассмотреть на примере атрибута width (ширина). При кодировании таблицы можно задать ее ширину, равную, например, 595 пикселям. Соответствующая запись выглядит следующим образом: <TABLE width="595">

Другим интересным примером служит значение, сопровождающее атрибут alt, который появляется в тегах изображений или объектов. Он предлагает описательное определение изображения или объекта тем пользователям, которые не могут или не хотят видеть само изображение или объект: <IMG src="images/molly.gif" alt="фотография">

В этом случае значение, присвоенное атрибуту alt, на самом деле является описанием изображения. В данном примере, кроме того, видно, каким образом тег может иметь несколько атрибутов.

Перед всеми значениями стоит символ = (знак равенства), а само значение находится в кавычках. Это общепринятый способ указания значений в строке HTML, за исключением шестнадцатеричных значений, где перед алфавитно-цифровым значением ставится символ #.

В HTML имеется подмножество, которое называется "множеством специальных символов". Это синтаксис HTML, с помощью которого создаются символы, обозначающие знаки препинания, и символы, необходимые для форматирования содержимого страницы.

Интересно, что во многих программах WYSIWYG для вызова знаков препинания, таких, как круглые и иные скобки или кавычки, всегда используются специальные символы. Однако многие просто вводят знаки препинания вручную. Обычно у браузеров не возникает проблем с интерпретацией ASCII-символов.

Специальные символы ничем не похожи на стандартный тег HTML. Примером служит символ авторского права, который можно закодировать как ©

Знак & обозначает начало специального символа, а точка с запятой (;) – его конец. Таким образом, браузеру понятно, что следует не показывать на экране слово "copy", а интерпретировать весь фрагмент как единый символ авторского права ©.

Иногда хочется показать на HTML-странице примеры кода HTML. Сделать это можно только с использованием специальных символов, иначе код HTML будет интерпретироваться буквально. Если ввести <IMG src= "dude.gif ">, то браузер эту текстовую информацию не покажет, а интерпретирует тег HTML и начнет поиск файла изображения с именем dude. gif.

Чтобы увидеть применяемый синтаксис, следует использовать знаки "меньше" и "больше" как специальные символы, и тогда вместо HTML будет показана текстовая строка: < IMG src="dude.gif" >

Набор специальных символов очень большой. Иногда один и тот же символ можно представить несколькими кодами.

2.2. СТРУКТУРА ДОКУМЕНТА HTML

Вся HTML-страница заключается в пару тегов

<HTML>

</HTML>

Между парой тегов <HTML> и </HTML> располагается сам документ. Документ может состоять из двух разделов – раздела заголовка (начинающийся тегом <HEAD>) и раздела содержательной части документа (начинающийся тегом <BODY>).

Раздел документа HEAD определяет его заголовок и не является обязательным тегом, однако хорошо составленный заголовок может быть весьма полезен. Задачей заголовка является представление необходимой информации для программы, интерпретирующей документ. Теги, находящиеся внутри раздела HEAD (кроме названия документа, описываемого с помощью тега <TITLE>), не отображаются на экране.

Раздел заголовка открывается тегом <HEAD>. Закрывающий тег </HEAD> показывает конец этого раздела. Между упомянутыми тегами располагаются остальные теги раздела заголовка.

В заголовок документа входит следующая информация.

– Название страницы. <TITLE> и соответствующий ему тег </TITLE> позволяют определить название страницы. Оно не появляется в теле HTML-страницы, т.е. при просмотре в браузере на главном экране эта информация не видна. Это название будет выведено в строке заголовка браузера. В элемент TITLE нельзя добавить никаких других тегов HTML, но можно использовать специальные символы.

– Сценарии. Любой сценарий, выполняемый на странице, например написанный на языке JavaScript, встраивается в заголовок документа.

– Стиль. Для тех разработчиков, которым нужно добавить в свои HTML-страницы элементы управления и стили, есть возможность встроить в заголовок каскадные листы стилей или связать их с Web-страницей. Эти данные также размещают в заголовке документа HTML.

– Метаинформация. Тег META известен своими разнообразными возможностями и мощью, которые позволяют выполнять множество "мыслительных" процессов, например, указывать автора документа, ключевые слова или специальные действия.

Это наиболее популярный элемент разметки заголовка, более распространен только элемент TITLE. Такое положение дел объясняется назначением данного элемента разметки. META содержит управляющую информацию, которую браузер использует для правильного отображения и обработки содержания тела документа.

Впервые тег META был задействован при принудительной перезагрузке документа браузером через заголовок HTTP-сообщения. В заголовке HTTP-сообщения можно указать оператор refresh. Время, заданное как параметр этого оператора, определяет интервал в секундах, после которого браузер загружает документ, определенный атрибутом URL данного оператора. Впервые этот механизм был реализован на сервере CERN, но наибольшую популярность приобрел при использовании сервера WN (Web-сервер, который был разработан для платформы Unix (Linux).

В теге META подобный механизм реализуется следующим образом:

```
<META HTTP-EQUIV="Refresh" CONTENT="1;
    URL=refresh.htm">
```

В данном случае через одну секунду после загрузки документа браузер должен инициировать загрузку страницы refresh.htm.

Используя этот механизм, можно построить автоматически перезагружаемую последовательность страниц. Для этого в заголовке каждой страницы из данной последовательности следует разместить соответствующий тег META.

```
<META HTTP-EQUIV="Refresh" CONTENT="1;
    URL=refreshX.htm">
```

Заглавная буква "X" в слове "refreshX.htm" – это цифра номера кадра. На странице нулевого кадра в этом месте следует указать на первый кадр (refresh1.htm), на странице первого кадра – на второй (refresh2.htm) и т.д.

В Windows 95 и Windows NT 4.0 с поддержкой таблиц UNICODE появилась возможность указывать тип кодировки документа – CHARSET. Для перекодировки на стороне клиента в заголовок документа необходимо включить META-тег следующего вида:

<META HTTP-EQUIV="Content-type"
CONTENT="text/html;
CHARSET=windows-1251">

Кроме Content-type, можно указать и другие операторы. Например, запретить кэширование документа. Необходимость в этом возникает при частом обновлении документа. Для запрета кэширования достаточно вставить в заголовок META-тег вида:

<META HTTP-EQUIV="Cache-Control"
CONTENT="no-cache">

С появлением роботов поисковых машин на META-тег была возложена еще одна функция – описание поискового образа документа. Собственно, для описания документа используются два META-тега. Один определяет список ключевых слов, а второй – реферат (краткое содержание документа), который отображается в качестве пояснения к ссылке на документ в отчете поисковой машины о выполненном запросе. Тег TITLE здесь также используется в качестве названия документа.

<TITLE>Основы Web-технологий</TITLE>
<META NAME="description"
http-equiv="description" content="Учебный курс Основы Web-технологий.
Тема: Заголовок HTML-документа. Элемент разметки META. Дается краткое описание основных способов применения тега META в заголовке HTML-документа. Рассматриваются управление HTTP-обменом и индексирование документов.">
<META NAME="keywords" HTTP-EQUIV="keywords"
content="учебный курс; Web-технология; web; технология; HTML; язык гипертекстовой разметки; заголовок HTML-документа; заголовок; HTML; документ; тег; META; элемент; HEAD; пример; разметка; методика">

При индексировании такого документа содержимое тега TITLE и атрибутов content тегов META после фильтрации попадет в индекс поисковой машины и может быть использовано для составления запросов. Процесс фильтрации отбракует так называемые stop-слова и общие слова. Они не попадут в индекс поисковой машины. В частности, будут отбракованы предлоги.

META-тегом пользуются и программы подготовки документов. Они размещают в нем свой идентификатор. В общем случае тег META выглядит следующим образом:

<META [name=имя] [HTTP-EQUIV=имя_HTTP-оператора]
content=текст>

Практика показывает, что при индексировании можно указывать одновременно и атрибут name, и атрибут HTTP-EQUIV с одинаковыми значениями. Это связано с тем, что одни роботы индексирования анализируют содержание *META*-элемента по атрибуту name, а другие – по атрибуту HTTP-EQUIV.

В теге <TITLE> следует указать название своей страницы. Оно должно быть простым, но ясным, описывающим содержимое страницы. Эта информация появится в строке заголовка браузера.

В теле документа HTML находится вся та информация, которая будет предлагаться для просмотра. Начинается тело страницы тегом <BODY>, заканчивается – </BODY>.

Теперь мы можем записать общую структуру страницы в таком виде:

<HTML>
<HEAD>
заголовок
</HEAD>
<BODY>
тело страницы
</BODY>
</HTML>

После создания оболочки одной из первых задач будет ввод информации в тело страницы. Это может быть:

– **Текст.** Текстовое содержимое узла находится в теле. Для представления информации используется форматирование, чтобы текст был удобочитаем и визуально доступен для посетителей узла.

– Изображения. Можно использовать графику для указания тематики узла, фотографию, дополняющую текст, или набор навигационных кнопок. В любом случае изображения являются важной частью информации, находящейся в теле документа.

– Ссылки. Ссылки дают возможность пользователям как перемещаться по узлу, так и переходить из него в другие места Web. Ссылки всегда размещаются в теле страницы.

– Мультимедиа и специальные программы. Для управления Shockwave, Flash, апплетами Java и даже встроенным видео используется код, размещаемый в теле документа HTML.

Иногда, особенно при создании объемных документов, возникает необходимость разделить страницу на несколько областей (в частности, для удобства чтения). Формальных методов разделения тела с помощью стандартных тегов на нужные разработчику разделы не существует. Однако есть очень простой способ помочь себе сделать это в области BODY и фактически во всем документе HTML.

Эта задача решается с помощью специального тега комментария. Он отличается от всех остальных тегов HTML тем, что не придерживается ни одного из правил синтаксиса.

Вот самый простой комментарий: <!– – – –>.

Тег комментария с текстом выглядит следующим образом: <!– – начало информации по авторскому праву – –>

Теги комментариев никогда не выводятся на HTML-странице, даже если они находятся в теле страницы.

Комментарии можно использовать везде, где есть смысл разделить на части и пометить документ HTML. Кроме того, комментарии можно использовать, чтобы добавить в страницу необходимые, но скрытые от пользователя сведения, такие,

как имя автора, контактные данные и т.п. Комментарии могут размещаться на нескольких строках, с открывающим тегом на первой и закрывающим на любой нужной вам строке.

2.3. ФОРМАТИРОВАНИЕ И ВЫРАВНИВАНИЕ ТЕКСТА

Форматирование текста – это самая простая и одновременно самая мощная часть HTML. Ведь в конечном итоге HTML был разработан для того, чтобы форматировать текстовые документы и делать их доступными в Интернете.

Большинство концепций форматирования текста просты: стандартные заголовки, разрывы, абзацы и текстовые стили, такие, как полужирный, курсив и подчеркивание.

Для форматирования текста можно использовать ряд тегов.

- <H1>...</H1>-<H6>...</H6>. Диапазон тегов заголовков применяется для заголовков и подзаголовков содержимого.
-
. Тег разрыва, который равнозначен одному возврату каретки.
- <P>. Тег абзаца используется для обозначения абзаца.
- <P>...</P>. Способ обозначения абзаца с помощью открытия/закрытия.
- <PRE>...</PRE>. Тег предварительного форматированного текста.
- <B>... </B>. Тег для задания полужирного начертания текста.
- <I>...</I>. Тег для задания курсива в тексте.
- <U>...</U>. Тег для подчеркивания текста.

2.3.1. Работа с заголовками

Заголовки HTML позволяют объявить определенные области документа, дав им название. Тег заголовка представляет собой алфавитно-цифровую комбинацию "H" и числового значения от 1 до 6, где 1 является наибольшим, а 6 – наименьшим уровнем заголовка. Теги заголовков работают в обратном порядке, т.е. наименьшее значение 1 создает наибольший видимый заголовок.

Теги заголовка просто окружают текст, который требуется использовать в виде заголовка соответствующего уровня:

<H1>Полезность упражнений для здоровья</H1>

Заголовки на странице автоматически выравниваются по левому краю, поэтому если вы их хотите выровнять по центру или по правому краю, то для этого следует использовать метод выравнивания, о котором речь пойдет чуть позже.

2.3.2. Вставка абзацев и пустых строк

Используя различные возможности их тегов, абзацы можно разбивать на отдельные видимые блоки текста. Тег абзаца форматирует эти блоки с помощью двух символов возврата каретки в том месте, где должен быть конец абзаца.

Есть два распространенных способа работы с тегами абзацев. Первый – простое использование тега <P> перед началом абзаца.

```
<HTML> <HEAD>
<TITLE>Польза от упражнений</ TITLE > </HEAD>
<BODY>
<H1>Полезность упражнений для здоровья</H1>
<P>Физические упражнения – прекрасный способ поправить здоровье. Медицинские исследования показывают, что с помощью упражнений можно укрепить сердце и легкие, уменьшить кровяное давление и поддерживать нормальный вес.
<P>Упражнения могут помочь улучшить настроение. Люди, занимающиеся упражнениями, имеют большее чувство собственного достоинства, больше решительности и в целом более положительный взгляд на жизнь.
</BODY>
</HTML>
```

Результат выполнения этого кода представлен на рис. 1.

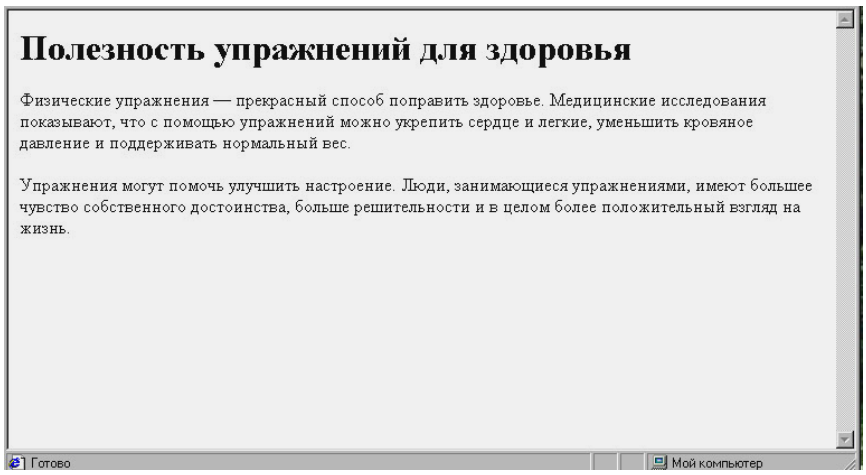

Рис. 1

Этот способ ясный и четкий, однако альтернативный способ, когда в начале и в конце абзаца используются соответственно теги открытия <P> и закрытия </P>, имеет дополнительное преимущество, состоящее в том, что к этому абзацу можно применять атрибуты.

Открытие с последующим закрытием становится особенно полезным, когда требуется выровнять текст или добавить стили с помощью листа стилей.

Иногда необходимо, чтобы при просмотре в браузере текст начинался с новой строки. Этого можно добиться с помощью тега разрыва
. Прекрасным примером того, где это можно делать, является кодирование адреса:

03142, Россия, Москва
ул. Кржижановского, 3
Соколовой Р.Ф.

Каждый тег разрыва вызывает переход к новой строке, не вставляя при этом промежуточные строки.

Многие стремятся использовать теги абзацев и разрывов для создания пустых строк между абзацами (и даже изображениями или объектами). Хотя это и неверный путь, но, чтобы избежать проблем, необходимо понимать, как браузеры работают с этими тегами.

Применяя тег абзаца для получения пустой строки, очень важно пользоваться методом единичного <P>. Как уже отмечалось, такой <P> вызывает **два** возврата каретки. Один из них вставляется в конце последней строки (как и в случае тега разрыва), а другой – в следующей строке. Поэтому если вам нужна одна пустая строка, то ее можно получить с помощью тега <P>. Тег абзаца можно считать равным двум тегам разрыва:
1 <P> = 2

Однако таким способом нельзя получить несколько пустых строк. Браузеры почти всегда игнорируют все остальные теги, стоящие после первого. Однако внутри абзаца можно вставлять разрывы. Чтобы получить пустые строки, просто вставьте нужное количество тегов разрыва после первого тега абзаца <P>

.

2.3.3. Применение текстовых стилей

Помимо обычных методов форматирования текстовой информации, таких, как создание библиографических ссылок, авторам и дизайнерам часто требуется привлечь особое внимание к определенной информации внутри текстового документа.

Для реализации этих возможностей форматирования в HTML можно использовать три главных текстовых стиля: полужирный, курсив и подчеркивание.

Тег полужирного стиля <B> и закрывающий тег </B> просто размещаются вокруг части текста, которая должна быть выделена таким способом:

Выделение части текста <B>полужирным</B> шрифтом.

То же относится и к тегам курсива – открывающему <I> и закрывающему </I>:

Выделение части текста <I>курсивом</I>.

Таков же принцип работы и тега подчеркивания <U>. Использовать подчеркнутый текст следует осторожно, в первую очередь потому, что обычно подчеркиваются ссылки и пользователи могут ошибочно принять за ссылку подчеркнутый текст.

Возможно одновременное применение нескольких элементов. Например, ***жирный курсив*** получается вложением

<I> <B> текст </B> </I>

При этом важно, чтобы внутренние скобки были закрыты раньше, чем внешние. При пересечении последствия получаются непредсказуемыми.

Поэтому приведенный выше вариант является правильным, а вот такой:

<I> <B> текст </I> </B> – неправильным.

Имеются также теги, родственные полужирному и курсиву и дающие те же результаты. Это теги <STRONG>.. .</STRONG> и <EM>.. .</EM>. Они часто используются в приложениях WYSIWYG и редко профессионалами, кодирующими вручную.

2.3.4. Использование предварительно отформатированного текста

Бывают случаи, когда в HTML-документ необходимо включить текст, уже имеющий форматирование, выполненное традиционным способом при помощи символов перевода строки, необходимого количества пробелов, символов табуляции и т.д. Для решения таких задач предусмотрен специальный тег <PRE>, определяющий предварительно форматированный (преформатированный) текст.

Текст, размеченный тегом <PRE>, будет отображаться в таком виде, как он выглядит в обычном текстовом редакторе. Для отображения всегда будет использоваться моноширинный шрифт. При этом вы сможете в большей степени контролировать вывод документа программой просмотра, правда, за счет некоторой потери в гибкости.

Одним из вариантов использования этого тега являются таблицы, построенные без применения специальных тегов разметки таблиц. Другим важным применением является вывод на экран больших блоков программного кода (Java, C++ и т.п.), не позволяющий браузеру переформатировать их.

Текст внутри тега <PRE> может содержать элементы форматирования уровня текста. Недопустимо внутри преформатированного текста задавать элементы форматирования уровня блока, например, теги заголовков. Тег абзаца по логике ве-

щей также не должен встречаться внутри преформатированного текста, однако если встречается, то будет реализовывать переход на новую строку (без образования пустой).

2.3.5. Выравнивание текста

Способ вывода текста на экран очень важен с точки зрения читаемости и эстетики. Выравнивание играет большую роль, определяя, каким образом текст располагается на странице по отношению к другим объектам.

Обычно текст выравнивается по левому краю (как минимум в большинстве западных языков). Это означает, что он начинается с левого края страницы. По правому краю текст неровный. Этот край называется *зазубренным правым*.

Однако имеются и другие виды выравнивания. В более коротких разделах текст можно выравнивать по правому краю или по центру. Таким образом, можно не только выделить текст, но и разделить пространство, чтобы дать глазам отдохнуть от стандартного выравнивания.

Рассмотрим существующие типы выравнивания.

- По умолчанию. Когда для текста не указано никакое выравнивание, браузеры выбирают для него значение по умолчанию, т.е. стандартное выравнивание по левому краю с зазубренным правым краем.
- По левому краю (значение атрибута align="left"). Это то же самое, что и значение для браузера по умолчанию. Однако его можно использовать и в качестве явного значения выравнивания. Это особенно полезно, когда на странице используется и другой вид выравнивания и требуется сделать так, чтобы часть текста была выровнена по левому краю.
- По правому краю (значение атрибута align="right"). Означает выравнивание текста по правому краю с зазубренным левым. Выравнивание по правому краю является интересным специальным эффектом, но обычно его надо использовать для выделения, а не для стандартного основного текста.
- По центру (значение атрибута align="center"). Центрирование текста, как и выравнивание по правому краю, создает визуальное выделение. Однако не следует злоупотреблять этим выравниванием. Многие дизайнеры-новички часто используют центрированный текст, но чего они в действительности добиваются, так это свободного пространства и наглядной текстуры, так как центрированный текст выглядит более интересно. Однако читать такой текст долгое время будет утомительно.
- По ширине (значение атрибута align="justify"). Означает такое расположение текста, при котором оба его края выровнены.

Выравнивание текста можно осуществлять двумя способами.

1. Выравнивание текста с помощью элемента абзаца. Используется тег <P align="…"> … </P> . Но поскольку тег абзаца влияет только на свой абзац, то для форматирования всей страницы значение придется присваивать каждому открывающемуся тегу абзаца. Хотя этот прием корректно работает, он является довольно громоздким.
2. Использование тега <DIV align="…"> … </DIV>. Внутрь этого тега помещают весь текст, выравниваемый в соответствии со значением атрибута align.

2.4. СПИСКИ

Списки в HTML позволяют разделить информацию на логические последовательности элементов. Создаваемые на основе стилей форматирования текста, эти списки достаточно стабильны, так как поддерживаются браузерами с самых ранних этапов развития языка.

Общие теги списков:

- <UL>...</UL> – неупорядоченный, или *маркированный* список;
- <OL>... </OL> – упорядоченный, или *нумерованный* список;
- <LI> – тег элемента списка (закрывающий тег не требуется).

Списки не только помогают ясно изложить информацию, но и логически упорядочивают ее, позволяя вести читателей согласно заранее установленному порядку от одного элемента к другому. Благодаря спискам содержимое документов можно представить таким образом, чтобы быстро и в нужной последовательности ознакомить читателей с основными их идеями.

Еще одна важная особенность списков – возможность использования отступов, которые позволяют разделять информацию. Таким образом, внимание посетителя узла сосредоточивается на необходимой информации, а кроме того, появляется малозаметный, но важный элемент дизайна: плавное течение текста, а не строгое ограничение части пространства. Списки, таким образом, делают документ более логичным, организованным и наглядным.

Чаще всего используются маркированные списки. В них перед каждым элементом вместо числовых значений упорядоченного списка ставятся некоторые символы. Для стандартного невложенного маркированного списка символом по умолчанию является диск. Неупорядоченные списки начинаются и заканчиваются тегами <UL> и </UL>. Между тегами можно разместить информацию. Ею чаще всего являются некоторые неупорядоченные элементы, перед каждым из которых стоит тег элемента списка <LI>, например:

```
<UL>
<LI>Ручка
<LI>Стакан с водой
<LI>Маленький желтый блокнот
</UL>
```

Эта информация появляется в виде однострочных элементов, перед каждым из которых стоит круглый маркер (рис. 2).

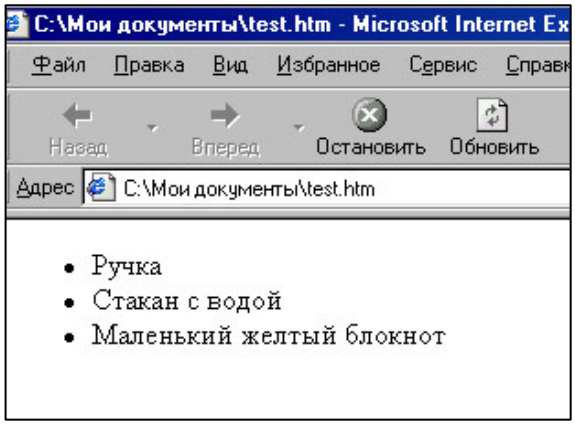

Рис. 2

Упорядоченные списки работают точно так же, как и маркированные, с одним исключением: вместо круглого маркера, выводимого на экран тегом строки элемента, используются последовательные числовые значения.

Начинаем с тега

<OL>

</OL>

Упорядоченные списки используются там, где больше подходит числовая нумерация, а не просто точки-маркеры.

<P>На нашем узле есть следующие произведения Чапека

<OL>

<LI> Редкий ковер

<LI> Истории о взломщике и поджигателе

<LI> История дирижера Калины

<LI> Смерть барона Гайдары

< LI > Похождения брачного афериста

< LI > Взломщик-поэт

<LI> Дело господина Гавлены

<LI> Игла

</OL>

Полученный в результате нумерованный список можно увидеть на рис. 3.

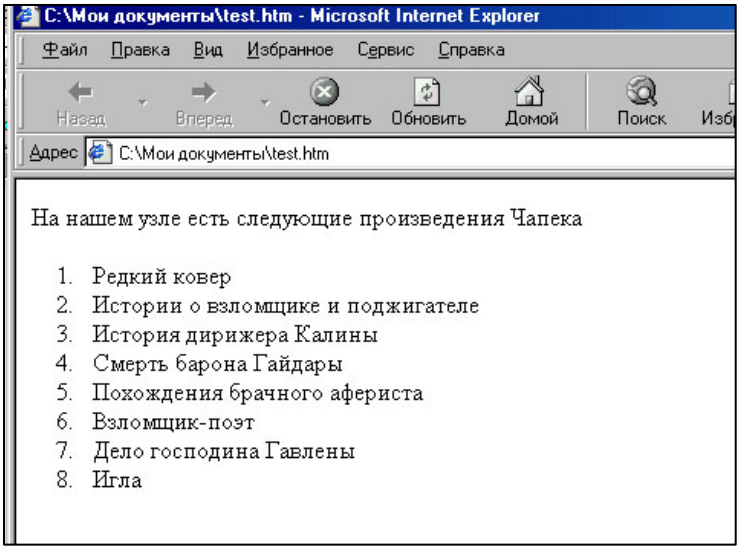

Рис. 3

Существует ряд приемов, позволяющих более эффективно манипулировать списками и их атрибутами. К ним относятся вложенные списки и изменение атрибутов списков.

Вложение – это размещение одного тега в другом. Этот процесс можно сравнить с матрешкой.

Попробуем создать неупорядоченный список с одним уровнем вложения.

<UL>

<LI> Кир Булычев

<LI> Братья Стругацкие

<LI> Александр Беляев

</UL>

Теперь **под** элементом списка вводим теги другого неупорядоченного списка

```
<UL>
<LI> Кир Булычев
        <UL>
        </UL>
<LI> Братья Стругацкие
        <UL>
        </UL>
<LI> Александр Беляев
        <UL>
        </UL>
</UL>
```

Заключительный этап – вводим несколько элементов списка во вложенные теги

```
<UL>
<LI> Кир Булычев
        <UL>
            <LI> Агент КФ
            <LI> Подземелье ведьм
        </UL>
<LI> Братья Стругацкие
        <UL>
            <LI> Хромая судьба
            <LI> Град обреченный
        </UL>
<LI> Александр Беляев
        <UL>
            <LI> Голова профессора Доуэля
            <LI> Человек-амфибия
        </UL>
</UL>
```

Результат, полученный при выполнении этого кода, показан на рис. 4. Обратите внимание, что у вложенного списка маркер отличается от маркера основного уровня.

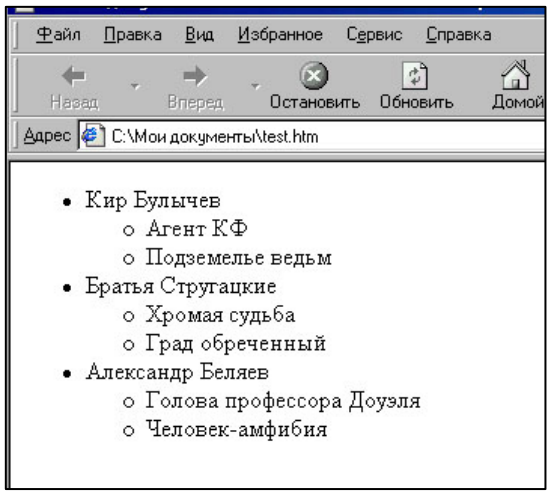

Рис. 4

Возможность контролировать порядок или внешний вид элементов списка обеспечивается с помощью нескольких атрибутов списков, включая value и type.

Если требуется продолжить упорядоченный список с некоторого числа, то к его элементу можно добавить значение номера с помощью атрибута value.

```
<OL>
<LI value="30"> Это 30-й элемент списка
<LI>Это 31-й элемент
<LI>И так далее
</OL>
```

Можно изменить внешний вид маркеров списка с помощью атрибута type в элементе списка, например:

```
<UL>
<LI type="disc">Маркер-диск
<LI type="circle"> Маркер-окружность
 <LI type="square"> Квадратный маркер
</UL>
```

На рис. 5 эти атрибуты показаны в работе.

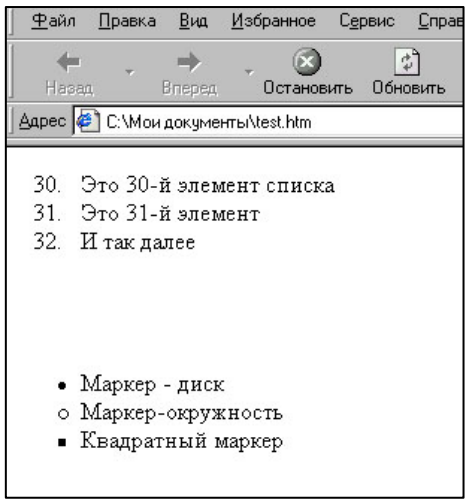

Рис. 5

В качестве маркеров списка можно использовать графические изображения, что широко применяется для создания привлекательных, красиво оформленных HTML-документов. На самом деле такая возможность не предоставляется непосредственно языком HTML, а реализуется несколько искусственно. Это вовсе не означает, что так делать не рекомендуется или предосудительно, а лишь означает, что здесь не будут применяться никакие специальные языковые конструкции HTML.

Чтобы понять идею, необходимо разобраться в механизме реализации списков на HTML-страницах. Оказывается, что тег списка <UL> (как, впрочем, и теги списков других типов) выполняет единственную задачу – указывает браузеру, что вся информация, располагаемая после данного тега должна отображаться со сдвигом вправо (отступом) на некоторую величину. Теги <LI>, указывающие на отдельные элементы списка, обеспечивают вывод стандартных маркеров элементов списка.

Если же нам требуется построить список с графическими маркерами, то можно вообще обойтись без тегов <LI>. Достаточно будет перед каждым элементом списка вставить желаемое графическое изображение. Единственной задачей, которую нужно при этом решить, будет отделение элементов списка друг от друга. Для этого можно использовать теги абзаца <P> или принудительного перевода строки
. Заметим, что использование графики на HTML-страницах может значительно увеличить объем передаваемой информации. Однако, если для всех маркеров используется один и тот же файл, который будет передан только один раз, это увеличение крайне незначительно. Размеры файла, содержащего маленькое изображение, также крайне незначительны.

Списки определений, также называемые *словарями определений* специальных терминов, являются особенным видом списков. В отличие от других типов списков, каждый элемент списка определений всегда состоит из двух частей. В первой части элемента списка записывается определяемый термин, а во второй части – текст в форме словарной статьи, раскрывающий значение термина.

Списки определений задаются с помощью тега <DL> (Definition List). Внутри тега тегом <DT> (Definition Term) помечается определяемый термин, а тегом <DD> (Definition Description) – абзац с его определением. Для тегов <DT> и <DD> можно не записывать соответствующие закрывающие теги.

В общем, список определений записывается следующим образом:

<DL>
<DT>Термин
<DD> Определение термина
</DL>

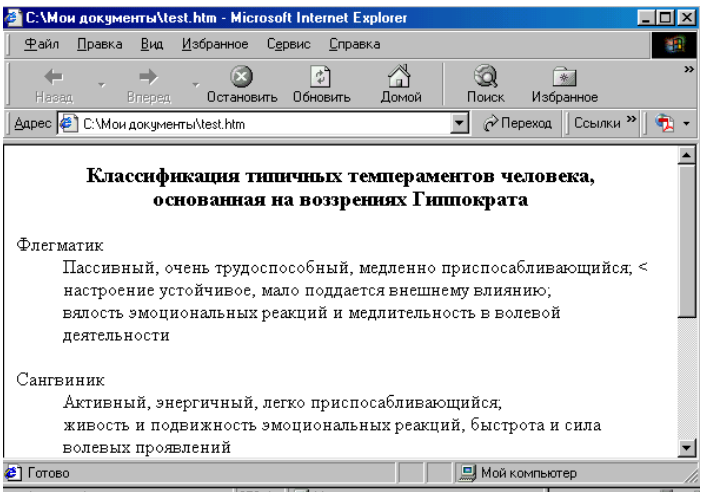

Рис. 6

В тексте после тега <DT> не могут использоваться элементы уровня блока. Как правило, текст определяемого термина должен располагаться в одной строке. Текст, содержащий определение термина, выводится, начиная со следующей строки (или через строку для некоторых браузеров) после определения термина с отступом вправо. В информации, помещенной после тега <DD>, могут располагаться элементы уровня блока. Отсюда следует, в частности, что списки определений могут быть вложенными.

Пример HTML-документа, в котором использован список определений:

```
<HTML>
<HEAD>
<TITLE>Пример списка определений</ TITLE>
</HEAD>
<BODY>
<DL>
<CENTER>
<H3>Классификация типичных темпераментов человека, <BR> основанная на воззрениях Гиппократа</H3>
</CENTER>
<DT>Флегматик
<DD>Пассивный, очень трудоспособный, медленно приспосабливающийся; <BR> настроение устойчивое, мало поддается внешнему влиянию; <BR> вялость эмоциональных реакций и медлительность в волевой деятельности <BR><BR>
<DT>Сангвиник
<DD>Активный, энергичный, легко приспосабливающийся; <BR> живость и подвижность эмоциональных реакций, быстрота и сила волевых проявлений <BR><BR>
<DT>Холерик
<DD>Активный, очень энергичный, настойчивый; <BR> порывистость и сила эмоциональных реакций, бурные волевые проявления <BR><BR>
<DT>Меланхолик
<DD>Пассивный, легко  утомляющийся, тяжело приспосабливающийся; <BR> слабость волевых проявлений и преобладание подавленного  настроения, неуверенность в себе
</DL>
</BODY>
</HTML>
```

Отображение приведенного HTML-документа в браузере показано на рис. 6.

2.5. СВЯЗЫВАНИЕ СТРАНИЦ

Связывание – вот в чем сущность Web. Связывание позволяет выйти за пределы одного документа и получить доступ не только к другим, относящимся к нему документам, но и ко всему богатству связанных с документом идей, а также к людям, имеющим к нему отношение.

Первоначально названный методом *гиперссылок* технический прием связи с другими документами в настоящее время позволяет работать не только с текстовыми ссылками. В качестве ссылок используется множество объектов и средств мультимедиа. Чтобы отразить этот аспект, был введен новый термин – *гипермедиа.*

Хотя синтаксис HTML для операции связывания достаточно прост, есть ряд деталей, которые следует обязательно знать, чтобы сполна использовать потенциал Web и возможности, предоставляемые связыванием.

Эти вопросы включают в себя работу с тегом якоря, использование относительных и абсолютных ссылок, а также управление специальными ссылками, например связыванием одного места на странице с другим или с электронной почтой.

Если сущность Web заключается в связывании, то сущность связывания – в HTML-теге A, т.е. в его якоре (или, как его еще называют, элементе привязки).

Тег якоря является тем компонентом, который позволяет одному HTML-документу связываться с другим.

Элемент привязки ограничен открывающим тегом <A> и закрывающим </A>. Чтобы тег якоря работал как положено, у него должны быть атрибуты и значения. Самым главным, простым и распространенным атрибутом является href, или *гипертекстовая ссылка.* После него вводят значение, чаще всего представляющее собой URL.

Вот как выглядит использование URL ТГТУ:

```
<A href="http://www.tstu.ru">
```

Любой текст или объект, находящийся между этой строкой и закрывающим тегом, будет представлять собой ссылку, при щелчке на которой вы попадаете с текущей страницы на указанную тегом:

Чтобы попасть на начальную страницу сайта ТГТУ, <A href="http://www. tstu.ru"> щелкните здесь.</A>

Эта ссылка (ее внешний вид показан на рис. 7) перенесет вас на Web-узел ТГТУ. Атрибут href может указывать на ресурс Internet, файл на локальном диске или метку внутри текущей страницы. Текст, расположенный внутри элемента A, представляет собой видимую часть гиперссылки. Именно на нем должен щелкнуть пользователь, чтобы осуществить переход. Браузер выделяет этот фрагмент цветом, а после использования гиперссылки меняет цвет, чтобы обеспечить подсказку.

Теги якоря создают ссылки двух видов: *абсолютные* и *относительные.*

Рассмотренный только что пример, где в качестве значения использовался полный URL, называется *абсолютной* ссылкой. Это означает, что вы использовали *абсолютно весь* Web-адрес, а не его часть. В ссылку включена информация и о протоколе, и о домене. Эти данные позволят вам перейти на начальную страницу по умолчанию этого Web-узла.

Абсолютные ссылки важны при задании в элементах привязки адресов на узлах, находящихся на других серверах.

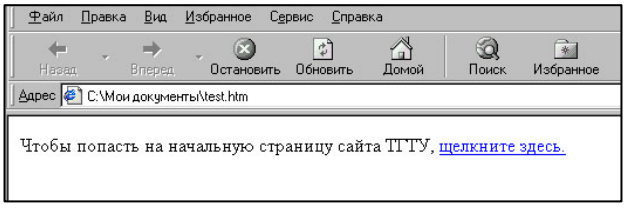

Рис. 7

Относительные ссылки позволяют связываться с файлами, находящимися по тому же адресу, т.е. на том же сервере. При создании ссылки с одной страницы на другую в пределах одного и того же узла, когда обе страницы находятся в одном и том же каталоге, все, что нужно сделать, – это присвоить в качестве значения *гипертекстовой ссылки* имя файла:

<A href="test.html"</A>

Задача несколько усложняется, когда необходимо создать ссылку на документ на том же сервере, но в другом каталоге. Например, если HTML-страница находится в главной папке, но имеется еще подпапка Jo, в которой резмещен файл jo.html. Тогда в гипертекстовой ссылке надо указать относительный путь к этому файлу:

<A href="jo/jo.html"> Новая страница</A>

Теперь браузеру известно, что поиск следует вести в каталоге jo, а не в том, где находится первоначальный документ.

К файлу всегда надо указывать точный путь из первоначальной Web-страницы. Если бы в каталоге jo была подпапка stories и требовалось бы установить ссылку из первого документа в главной папке, то в ссылке пришлось бы указать полный путь к тому файлу, который требуется загружать в браузере. Допустим, будем загружать файл travels.html. Синтаксис в этом случае должен выглядеть таким образом:

<A href="jo/stories/travels.html">Страница о приключениях </A>

Чтобы перейти из папки в ее родительскую папку, для относительных ссылок используются ".." (две точки подряд). Таким образом, для перехода из jo.html к index.html, находящемуся в главной папке, необходимо создать следующий код:

<A href="../index.html">Назад, на начальную страницу</A>

С его помощью можно попасть в верхний, или *корневой* каталог, где находится файл index. html.

Весьма полезным методом перемещения являются внутренние ссылки. Данный вид ссылок указывает текст, который располагается не на другой странице, а на той же, где и ссылки. В этом случае месту на странице, к которому происходит переход по внутренней ссылке, должно быть присвоено имя. Это называется меткой, или именованным элементом привязки:

<A name="Метка"> Текст </A>

Для перехода к именованному элементу привязки необходимо создать ссылку следующего вида:

<A href="#Метка"> Текст ссылки </A>

Когда пользователь щелкнет на тексте ссылки, он переместится на именованный элемент привязки.

В качестве примера внутренних ссылок рассмотрим следующий код:

```
<P>На этой странице вы найдете произведения
<A href="#Bulychev">Кира Булычева</A>
<A href="#Stugatski">братьев Стругацких</A>
<A href="#Belyaev">Александра Беляева</A> и других

<LI> <A name=Bulychev> Кир Булычев </A>
        <UL>
        <LI> Агент КФ
        <LI> Подземелье ведьм
        </UL>
<LI> <A name=Stugatski> Братья Стругацкие </A>
        <UL>
            <LI> Хромая судьба
            <LI> Град обреченный
        </UL>
<LI><A name=Belyaev > Александр Беляев </A>
        <UL>
            <LI> Голова профессора Доуэля
            <LI> Человек-амфибия
        </UL>
</UL>
```

Результат выполнения этого кода представлен на рис. 8.

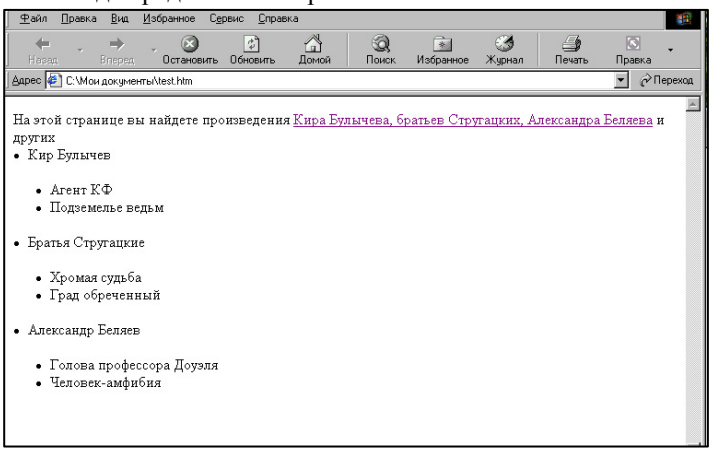

Рис. 8

Когда гиперссылка используется для указания адреса электронной почты, ее выбор обеспечивает не переход к новому документу, а запуск диалога для отправки сообщения указанному адресату. Обычно такую ссылку размещают в конце страницы для обеспечения связи с Web-мастером или автором страницы. Пример такой ссылки:

<A href="mailto:alexs@pccenter.ru"></A>

Ниже приведены примеры ссылок на различные файлы по разным протоколам:

- Ссылка на HTML файл по протоколу http

<A href="http://www.arcus.lv/dimas/page1.html">Пример</A>

- Ссылка на ZIP файл по протоколу http

<A href="http://www.arcus.lv/dimas/test.zip">Пример</A>

- Ссылка на TXT файл по протоколу http

<A href="http://www.arcus.lv/dimas/test.txt">Пример</A>

- Ссылка на EXE файл по протоколу FTP

<A href="ftp://ftp.sausage.com/pub/hotdog/hotdog6/hotdog6install.exe"> Пример</A>

- Ссылка на e-mail, при помощи протокола mailto

<A href="mailto:name@domen.ru">Пример</A>

- Ссылка на группу новостей, при помощи протокола news

<A href="news:comp.infosystems.www.announce">Пример</A>

Последняя ссылка будет работать, если только провайдер поддерживает доступ к телеконференциям.

2.6. ГРАФИКА НА WEB-СТРАНИЦАХ

2.6.1. Форматы графических файлов

Существует много способов описания графической информации, соответственно, имеется значительное количество форматов хранения графических файлов, – порядка нескольких десятков.

Все форматы хранения графической информации можно разделить на два типа: векторный и растровый.

Файлы векторной графики содержат математические данные о том, как перерисовать изображение с помощью отрезков прямых (векторов) при выводе его на экран. Процесс вывода требует дополнительной обработки, но такое представление графической информации имеет важное преимущество: масштаб изображения может быть изменен без потери качества, так как не существует фиксированной связи между тем, как он определен в файле и выводом точек на экран. При масштабировании растровой графики обычно происходит потеря разрешения, что ухудшает качество изображения.

Векторная графика, как правило, употребляется для изображений с четкими геометрическими формами. В векторном виде хранится информация для некоторых типов шрифтов.

Растровая графика предполагает хранение данных о каждой точке изображения. Для отображения растровой графики не требуется сложных математических расчетов, достаточно лишь получить данные о каждой точке и отобразить их на экране.

На Web-страницах в подавляющем большинстве случаев используется растровая графика в двух форматах: GIF и JPG. Именно эти два формата непосредственно поддерживаются популярными браузерами, а для использования большинства других графических форматов потребуются специальные средства.

Формат BMP является стандартом MS Windows и поддерживается браузером Internet Explorer, однако его употребление не может быть рекомендовано, так как данный формат не поддерживает сжатие данных.

Иные графические форматы (кроме GIF и JPG) в HTML-документах на WWW-серверах практически не встречаются, хотя принципиально это возможно. Использование других форматов не рекомендуется по следующим причинам: во-первых, только для GIF и JPG осуществляется встроенная поддержка в большинстве браузеров, тогда как для иных файлов необходимо подключение внешних программ отображения, во-вторых, структура файлов GIF и JPG наиболее подходит для передачи данных по сети и является не зависимой от платформы.

Формат файла GIF (Graphics Interchange Format) первоначально был предложен корпорацией CompuServe Incorporated для передачи графических данных по сети. Из-за популярности в сети CompuServe формат GIF получил широкое распространение и в настоящее время поддерживается множеством программ работы с графикой, а также реализован на ряде плат-

форм. Популярность формата увеличивается за счет свободного распространения его спецификации и свободного использования. Поскольку изначально формат разрабатывался для передачи данных в потоке, а не как формат для хранения данных в файле, то его последовательная организация как нельзя более подходит для размещения графических изображений на WWW-серверах. К положительным качествам формата можно отнести небольшой размер файлов изображения, возможность хранения множественных изображений, внесение перекрывающего текста, прозрачность, отображение ряда изображений с задержкой, задание режимов восстановления предыдущего изображения, введение данных для специфических приложений. К недостаткам следует отнести ограниченное количество цветов (не более 256), реализованных в виде палитры 24-битовых цветов, отсутствие возможностей по хранению градаций серого и данных цветовой коррекции, хранению данных в моделях, отличных от RGB (например, CMYK или HSI). Хотя 256 цветов во многих случаях оказываются достаточными, сохранение фотореалистичных изображений в этом формате может привести к ухудшению цветовой гаммы картинки.

В GIF-файле определены два различных варианта хранения данных. В одном из них все строки изображения записываются подряд от начальной до конечной (построчное хранение – Noninterlaced). В другом варианте строки сохраняются в определенном порядке (хранение с чередованием строк – Interlaced). Для последнего варианта порядок хранения строго определен, а именно, строки изображения с чередованием размещаются в четыре прохода:

- каждая 8-я строка, начиная с 0-й;
- каждая 8-я строка, начиная с 4-й;
- каждая 4-я строка, начиная с 2-й;
- каждая 2-я строка, начиная с 1-й.

Вариант хранения изображения задается при его создании или сохранении после редактирования и является параметром самого файла изображения. В зависимости от варианта хранения выполняется и вывод изображения на экран – либо картинка разворачивается сверху вниз, либо она постепенно проявляется и улучшается от прохода к проходу. Очевидно, что браузеры графических файлов могут использовать свои варианты появления изображения на экране, применяя различные эффекты вне зависимости от схемы хранения, однако для этого необходимо первоначальное получение всего файла с последующей его обработкой для получения нужного эффекта. Для WWW-браузеров характерно отображение файлов в процессе их получения, что определяет однозначную связь между схемой хранения и процессом выдачи на экран.

На первом проходе заполняется 1/8 часть всех строк изображения. При чересстрочной схеме хранения уже после первого прохода можно увидеть контуры появляющегося изображения и, при необходимости, остановить дальнейшую загрузку, чтобы не загружать сеть перекачкой бесполезной для вас информации. При построчной схеме хранения, получив данные для 1/8 части всех строк, пользователь увидит верхнюю 1/8 часть изображения сразу в оригинальном качестве, по которой, скорее всего, не удастся определить содержание рисунка.

Формат файлов графических изображений JPG (JPEG) был разработан Объединенной группой экспертов в области фотографии (Joint Photographic Experts Group) как средство для хранения изображений, имеющих большую глубину цвета (24 бита на пиксель, что обеспечивает 16,7 млн. возможных цветов).

Не останавливаясь на деталях хранения информации в этом формате, отметим, что файлы JPG следует использовать, прежде всего, для хранения фотореалистичных изображений. Ограничение в 256 цветов, присущее GIF, может снизить качество изображения, что исключается при использовании JPG. Поскольку JPG основан на сжатии данных с потерями, учитывающими особенности восприятия изображения человеком, то без значительного ухудшения картинки можно обеспечить значительную степень сжатия и, как следствие, небольшой размер файла. Аналогичный файл GIF в большинстве случаев будет иметь больший размер.

Еще одним важным параметром файлов JPG является схема их хранения. Различают две схемы – обычная и прогрессивная (progressive). Прогрессивная схема хранения такова, что при выводе изображений создается впечатление постепенного проявления рисунка на экране со все большим уточнением отдельных деталей. Это напоминает проявление изображения при работе с чересстрочными файлами формата GIF, однако здесь уточнение производится не построчно, а, как правило, по прямоугольным областям размера 8×8 или более. При сохранении изображения в обычной форме его отображение будет выполняться путем разворачивания изображения сверху вниз. Из сказанного можно сделать вывод, что хранение изображений, предназначенных для загрузки по сети, лучше осуществлять в прогрессивной форме.

Недостатком JPG можно считать то, что при сжатии с его помощью изображений с четкими контурами и резкими переходами между цветами линии начинают заметно "расплываться". Если изображение содержит текст, то подобный эффект может возникнуть вокруг символов. По этой причине нецелесообразно использовать JPG-изображения для представления снимков экрана (например, при описании интерфейса программных продуктов).

Прозрачность в JPG-изображениях невозможна, поскольку сжатие приводит к некоторым изменениям структуры изображения.

В каких случаях предпочтительнее использование формата GIF, а в каких – JPG?

Формат GIF следует использовать для изображений, создаваемых программным путем или рисуемых вручную с помощью графических редакторов, например, графики, гистограммы, несложные рисунки и т.д. (так называемый line art). Ограничение формата – одновременное использование не более чем 256 цветов – для таких изображений в большинстве случаев не играет роли. Алгоритм сжатия, используемый в GIF-формате (LZW – алгоритм, названный по фамилиям Lempel-Ziv-Welch), выполняющий сжатие без потерь, обеспечивает точное восстановление изображения и для несложных рисунков достаточно хорошую степень сжатия.

Формат GIF лучше всего подходит для следующих типов изображений:

– изображений с ограниченным количеством используемых цветов;

– изображений, имеющих четкие границы и края, что свойственно большинству изображений типа меню, кнопок и графиков;

– изображений, в состав которых входит текст.

Формат JPG больше подходит для хранения следующих изображений:

– фотографий, полученных со сканера или цифровой камеры, а также фотореалистичных изображений, построенных на основе компьютерных расчетов (ray-tracing rendering);

– графики со сложным сочетанием цветов и оттенков;

– любого изображения, которое требует более 256 цветов.

Некоторые проблемы со свободным использованием формата GIF возникли в 1995 году. Алгоритм компрессии LZW имеет патент, который в настоящее время принадлежит компании Unisys Corporation. Хотя GIF-формат является свободно распространяемым и используемым, однако компания Unisys Corporation в 1995 году решила обязать всех коммерческих продавцов программных продуктов, использующих LZW-компрессию, лицензировать ее применение. Чтобы избежать уплаты гонорара вместо GIF-формата было решено создать альтернативный формат PNG (Portable Network Graphic, читается "пинг"), поддерживающий прозрачный цвет и чередование строк, однако не допускающий нескольких изображений в одном файле. Данный формат является самым "молодым" – его стандартизованная и одобренная W3C спецификация PNG 1.0 существует лишь с октября 1996 г. И, как свойственно всему новому, он активно вытесняет с просторов Интернета своих предшественников, в первую очередь – GIF. Это обусловлено следующими обстоятельствами:

- PNG является бесплатным форматом, и его применение в программных продуктах не требует лицензионных отчислений;
- PNG обеспечивает глубину цвета 24 бита;
- PNG использует эффективный алгоритм сжатия без потерь, основанный на предварительной фильтрации сжимаемой информации.

Кроме того, в отличие от GIF, в новом формате предусмотрено не два уровня прозрачности (прозрачный/непрозрачный), а 254, что позволяет создавать сложные многослойные изображения.

Кроме того, в формате PNG предусмотрена еще более хитрая схема хранения данных об отдельных точках изображения по сравнению с чересстрочной схемой формата GIF. Если для формата GIF при чересстрочной схеме хранения можно было увидеть контуры всего изображения после загрузки примерно 1/8 части данных, то для формата PNG то же самое можно сделать, загрузив 1/64 часть данных.

Как и GIF, формат PGN поддерживает чтение и вывод данных на экран по мере их поступления, однако по сравнению с GIF фрагменты изображения являются двумерными, т.е. изображение проявляется не построчно, а в виде небольших прямоугольников.

Диапазон возможных вариантов использования графики на Web-страницах очень широк и многообразен. Рассмотрим описание некоторых наиболее распространенных вариантов использования изображений.

2.6.2. Фоновые изображения

Цвет фона страницы задается атрибутом bgcolor="цвет" тега <BODY>.

В качестве цвета можно указать либо английское название одного из 16 цветов, либо шестнадцатеричное значение любого из более чем 16 миллионов цветов.

В языке HTML фиксированными названиями цветов являются:

BLACK	Черный	**OLIVE**	Оливковый
TEAL		**RED**	Красный
BLUE	Синий	**MAROON**	Коричневый
NAVY	Темно-синий	**GRAY**	Серый
LIME	Светло-зеленый	**FUCHSIA**	
GREEN	Зеленый	**PURPLE**	Фиолетовый
SILVER	Серый	**YELLOW**	Желтый
AQUA	Голубой	**WHITE**	Белый

Следует учитывать, что некоторые браузеры обрабатывают гораздо больше названий цветов.

Удобнее задать цвет в формате RGB – красный – зеленый – синий. Каждое из трех чисел задает интенсивность соответствующего цвета в относительных единицах от 0 до 255. В этой форме цвет имеет вид #RRGGBB, где RR соответствует интенсивности красного, GG – зеленого и BB – синего цвета.

Вместо цвета фона можно задать фоновую картинку с помощью атрибута background="имя-файла". В качестве формата фоновых изображений желательно использовать только форматы GIF и JPG.

В случае одновременного указания параметров bgcolor и background картинка перекрывает фоновый цвет, который частично виден только при использовании прозрачного GIF.

Фоновое изображение для HTML-документа всегда заполняет все окно просмотра. Если размер изображения меньше размеров окна просмотра, то оно будет размножено по принципу мозаики. Поэтому фоновые изображения должны создаваться так, чтобы при появлении на экране границы сшивки повторяющихся изображений были бы невидимы. Эта задача напоминает подбор рисунка при оклеивании обоями стен комнаты.

Обычно в качестве фонового берется небольшое изображение, для загрузки которого по сети не требуется значительного времени. Существуют огромные коллекции изображений (текстур), которые можно использовать при разработке своих собственных HTML-документов.

Другим часто используемым вариантом является фоновое изображение в виде бледного рельефного логотипа. Такая графика ясно идентифицирует сайт и не мешает восприятию материала.

Пример записи тега <BODY> с указанием фонового цвета и фонового изображения:

<BODY background=texture.gif bgcolor=gray>

Одновременное задание параметров BACKGROUND и BGCOLOR вовсе не обязательно. Любой из них, равно как и оба вместе, могут отсутствовать.

На первый взгляд может показаться, что указание фонового цвета излишне при задании фонового изображения. В действительности все наоборот. Можно рекомендовать всегда указывать цвет фона документа, если задается фоновое изображение. Дело в том, что при загрузке документа, прежде всего, отображается текстовая часть, а на следующем проходе будут загружаться изображения, в том числе и изображение, используемое в качестве фонового. До момента загрузки и отображения фонового изображения цвет фона документа будет определяться значением атрибута bgcolor или устанавливаться по умолчанию. Опыт работы с HTML-документами, получаемыми по сети, показывает, что до загрузки фонового изображения порой проходит достаточное количество времени, в течение которого пользователь знакомится с уже загруженным текстом. В какой-то момент проявляется фоновое изображение, изменяя гамму цветов документа. Чтобы предотвратить резкое изменение гаммы цветов, следует задавать значение цвета фона близким к цветам фонового изображения.

При выборе цвета фона и характера фонового изображения следует не забывать о необходимости контраста между цветом текста и фона. Неудачное соотношение цветов может затруднить чтение текста.

Есть еще причина, из-за которой задание цвета фона документа следует рекомендовать. Пользователь может отключить загрузку изображений. В этом случае фоновое изображение также не будет загружено.

Еще один вариант предпочтений пользователя исключит и выдачу фонового изображения, и указание цвета фона. Конкретный вариант настроек зависит от используемого браузера.

2.6.3. Иллюстрации

Для размещения на странице изображений чаще всего используется тег <IMG> без закрывающего тега. Для тега <IMG> требуется указать URL изображения в атрибуте src.

<IMG src="foto.jpg">

Изображения всегда размещаются в теле документа HTML.

У тега <IMG> существует целый ряд атрибутов, позволяющих управлять положением и внешним видом изображения на странице:

– width="x" – позволяет браузеру заранее установить ширину изображения;

– height="x" – используя этот атрибут высоты вместе с атрибутом width, браузер может заранее подготовить место для изображения на странице.

Значения этих атрибутов могут указываться как в пикселях, так и в процентах от размеров окна просмотра.

Значения атрибутов ширины и высоты изображения могут не совпадать с истинными размерами изображения. В этом случае браузеры автоматически при загрузке изображений выполняют его перемасштабирование. Неаккуратное задание атрибутов может привести к изменению пропорций рисунка и, как следствие, к его искажению.

Любой из этих двух атрибутов может быть опущен. Если задан только один из атрибутов, то при загрузке рисунка значение второго атрибута будет вычисляться автоматически из условий сохранения пропорций. Изменение размеров изображений при помощи задания атрибутов ширины и высоты может использоваться для просмотра иллюстраций в уменьшенном виде, однако такой подход не сокращает время загрузки изображения.

– border="x" – этот атрибут используется для определения ширины рамки вокруг рисунка. Чтобы графика выводилась без рамок, лучше всего явно задать для атрибута border значение 0:

<IMG src="foto.jpg" border="0">

Если вокруг изображения рамка нужна, то ее можно задать с помощью числового значения атрибута border:

<IMG src="foto.jpg" border="4>

– align="x" – с помощью этого атрибута можно выровнять изображение по горизонтали и вертикали. Чаще всего применяют выравнивание по горизонтали, используя следующие значения атрибута:

left (по левому краю)

right (по правому краю).

Однако этот атрибут можно использовать и для выравнивания по вертикали с помощью значений:

top – помещает изображение вдоль самой верхней части содержащей его строки;

middle – изображение выравнивается по средней или базовой линии содержащей его строки;

bottom – изображение выравнивается по нижней части содержащей его строки;

– alt="описание" – позволяет выводить на экран вместо изображения его описание; это описание отображается на экране браузера до тех пор, пока не произойдет загрузка изображения. Кроме того, этот атрибут позволяет выводить на экран элемент подсказки с описанием, когда указатель мыши располагается над изображением. Очень важный атрибут. Для пользователей, не работающих с графикой, атрибут alt дает великолепное средство описания того, что могло бы быть на этом месте при использовании графики (рис. 9). Кроме того, поскольку графика загружается значительно медленнее текста, описание рисунка – весьма удачный способ поддержать у посетителя интерес к загружаемому материалу. Современные браузеры будут также отображать альтернативный текст в качестве подсказки (tooltip) при помещении курсора мыши в область изображения.

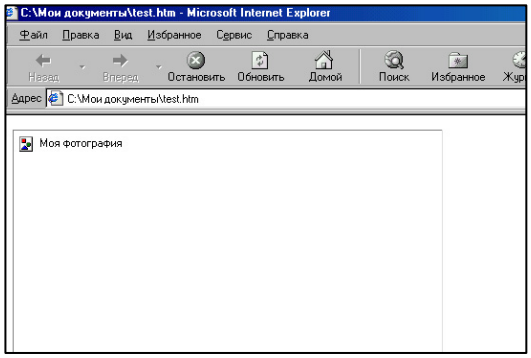

Рис. 9

– hspace="x"– свободное пространство по горизонтали; используется для добавления свободного места справа и слева от изображения. Устанавливается в виде числового значения

– vspace="x" – то же, что и hspace, но для свободного пространства по вертикали.

2.6.4. Использование миниатюрных версий изображений

На Web-страницах часто используются миниатюрные версии изображений (thumbnail) в качестве графических указателей ссылок на полноразмерные изображения. Изображения такого рода представляют собой уменьшенные копии оригинальных изображений, имеющие иногда также меньшую глубину цвета или представленные в оттенках серого цвета. Файлы с такими изображениями занимают значительно меньше места по сравнению с полноразмерными и поэтому гораздо быстрее грузятся. С помощью миниатюрных версий можно быстро просмотреть набор изображений, доступных для загрузки, и выбрать понравившееся.

2.6.5 Средства навигации

Изображения, как и текст, могут быть ссылками. Делать изображение ссылкой – распространенная практика и один из основных приемов подготовки навигации.

Для связывания изображений нужно поместить код изображения внутрь стандартного элемента связывания <A>, например:

<A href="index.html"><IMG src="images/home_botton.gif" alt=""На домашнюю страницу>
</A>

2.7. УПРАВЛЕНИЕ ЦВЕТОМ ТЕКСТА И ССЫЛОК

Самым мощным в HTML является тег BODY, определяющий тело документа. Его атрибуты позволяют управлять большинством визуальных возможностей Web-страниц: цветом и дизайном фона, цветом текста, цветами ссылок, размещаясь при этом внутри одного открывающегося тега <BODY>.

Задание цвета текстовой части документа задается атрибутами:

- text (цвет основного текста документа);
- link (цвет текста в ссылках);
- vlink (цвет ссылки, просмотренной ранее);
- alink (цвет ссылки, на которую в данный момент производится указание, т.е. этот цвет возникает в момент нажатия на клавишу мыши и исчезает после ее отпускания).

Способ задания цвета текстовой части ничем не отличается от задания цвета фона.

2.8. УПРАВЛЕНИЕ ШРИФТАМИ

Если рассмотренных возможностей по управлению шрифтами недостаточно, можно применить элемент <FONT>, имеющий вид:

<FONT color=цвет size=размер>
выделяемый текст
</FONT>

Слово color задает цвет букв, отменяя тем самым значение, указанное в BODY для выделенной части страницы. Способ задания цвета – как в BODY.

Слово size указывает размер шрифта – абсолютный в виде числа без знака, либо приращение по отношению к базовому размеру (обычно выбираемому пользователем браузера) в виде числа со знаком. Естественно, что + задает увеличение, а – уменьшение шрифта. Относительный размер предпочтительнее, поскольку автор заранее не знает ни возможностей монитора у читающего страницу, ни его предпочтений.

Настройки размеров шрифта, используемых по умолчанию, а также величины абсолютного изменения размеров шрифта зависят от браузеров. Принято считать, что размер "нормального" шрифта соответствует значению 3.

размер шрифта – <font size="число от 1 до 7">

текст размера 1
текст размера 2

текст размера 3
текст размера 4
текст размера 5
текст размера 6
текст размера 7

В дизайне обычно используются первые четыре размера. Все, что выше, занимает слишком много места, трудно читается и вообще некрасиво. Правда, иногда используют и большие размеры. Например, седьмым размером можно писать буквицу (первую букву начала главы).

Следует отметить, что при указании размеров шрифтов записи типа "2" и "+2" дают принципиально разный результат.

IE позволяет задавать начертание (гарнитуру) шрифта, которым текст будет выводиться на экран с помощью атрибута в теге <FONT face= [название шрифта]>, и браузер попытается отобразить текст тем альтернативным шрифтом, который указан. Если такого шрифта не будет найдено, то данное указание будет проигнорировано и будет использован шрифт, установленный по умолчанию.

Можно указать как один, так и несколько названий шрифтов, разделяя их запятыми. Это весьма полезное свойство, так как в разных системах могут быть почти идентичные шрифты, называющиеся по-разному. Другим важным качеством является задание предпочтения использования шрифтов. Список шрифтов просматривается слева направо. Если на компьютере пользователя нет шрифта, указанного в списке первым, то делается попытка найти следующий шрифт и т.д.

Рассмотрим следующий фрагмент кода:

```
<H1>Работа с тегом &lt;FONT&gt;.</H1>
<FONT face= "Verdana" size='5' color="#008000" >Шрифт этого текста "Verdana".</FONT><BR>
<FONT face = "Comic Sans MS" size =+2 color ="#FF0000">Шрифт этого текста "Comic Sans MS".</FONT><BR>
<FONT face = "Arial" size =+2 color ="#800000">Шрифт этого текста "Arial".</FONT><BR>
<FONT face = "Courier New" size =+2 color ="#0000FF">Шрифт этого текста "Courier New".</FONT><BR>
<FONT face = "Impact" size =+2 color ="#FF00FF">Шрифт этого текста "Impact".</FONT><BR>
```

Результат работы этого кода представлен на рис. 10.

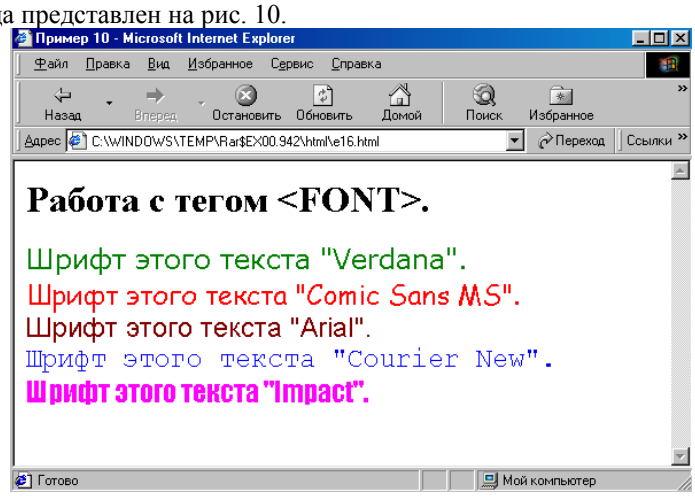

Рис. 10

Следует сделать замечание о типах шрифтов, используемых на страницах Web. Существует несколько типов шрифтов.

- Рубленый шрифт (без засечек) – по умолчанию Arial или Helvetica. Семейство обозначается *sans-serif*.
- Шрифт с засечками – это шрифты вроде Times New Roman, обозначается *serif*. Засечки придают горизонтальную устойчивость тексту, делая его более легким для чтения. Шрифт без засечек труден для чтения, особенно, если он мелкий. В этом случае необходимо увеличивать расстояние между строками.
- Моноширинный шрифт – типа Courier New, он имеет одинаковую ширину букв и похож на шрифт печатной машинки, обозначается *monospace*.
- Шрифт, имитирующий рукопись, обозначается *cursive*, например, French Script MT.
- Декоративный шрифт, обозначается *fantasy*, например, Braggadocio.

2.9. ТАБЛИЦЫ

Таблицы являются удобным и простым способом разделения частей страницы на горизонтальные и вертикальные фрагменты, выровненные между собой. Как и в обычных таблицах, части могут быть отделены или не отделены друг от друга рамкой из вертикальных и горизонтальных линий.

Очень часто на практике таблицы используются для размещения на страничке рисунков и текста в том виде, в котором страница будет выглядеть одинаково на экранах монитора с разным разрешением. Это достигается особым свойств таблиц растягивать длину и ширину ячейки по величине самого большого содержимого ячейки. Другими способами организовать так страничку, чтобы она одинаково выглядела для различных пользователей, довольно сложно. Кроме того, таблица, со-

стоящая из одной ячейки, может очень эффектно выделить фрагмент текста, на который следует обратить внимание читателя.

Таблица целиком заключается в пару тегов

<TABLE>

</TABLE>

Указание закрывающей скобки для таблицы, как и для всех ее составных частей, является очень важным. В случае незакрытой таблицы одни броузеры не отображают вообще ничего, другие искажают документ до неузнаваемости.

Клетки (ячейки) таблицы задаются в порядке горизонтальных рядов слева направо (рис. 11):

Ячейка 1	Ячейка 2	Ячейка 3
Ячейка 4	Ячейка 5	Ячейка 6

Рис. 11

Каждый горизонтальный ряд заключается в теги

<TR>

</TR>

а каждая ячейка внутри ряда – в теги

<TD>

</TD>

Таблица, приведенная на рис. 11, была задана следующим кодом:

```
<table border=1>
<tr><td>Ячейка 1</td>
<td>Ячейка 2</td>
<td>Ячейка 3</td></tr>
<tr><td>Ячейка 4</td>
<td>Ячейка 5</td>
<td>Ячейка 6</td></tr>
</table>
```

В этом примере для задания ширины линий рамки указан атрибут border. Число после ключевого слова задает ширину рамки в пикселях. Значение border =0 указывает на отсутствие линий рамки.

Тег <TABLE> может включать и другие атрибуты:

- ALIGN – выравнивание таблицы по отношению к краям документа. Допустимые значения: ALIGN='LEFT' (выравнивание влево), ALIGN='CENTER' (выравнивание по центру), ALIGN='RIGHT' (выравнивание вправо).
- WIDTH – ширина таблицы. Ее можно задать в пикселях (например, WIDTH=400) или в процентах от ширины страницы (например, WIDTH=80%).
- HEIGTH – высота таблицы. Ее также можно задать в пикселях или в процентах от ширины страницы.
- CELLSPACING – устанавливает расстояние между рамками ячеек таблицы в пикселах (например, CELLSPACING=2).
- CELLPADDING – устанавливает расстояние между рамкой ячейки и текстом в пикселах (например, CELLPADDING=10).
- BACKGROUND – задает URI фонового рисунка для таблицы.
- BORDERCOLOR – задает цвет рамки.
- BORDERCOLORDARK – задает цвет "затененной" части рамки.
- BORDERCOLORLIGHT – задает цвет "светлой" части рамки.

Таблица может иметь заголовок (<CAPTION> ... </CAPTION>), хотя заголовок не является обязательным (это название таблицы, примерно, как в книгах). Метка <CAPTION> может включать атрибут ALIGN. Допустимые значения: <CAPTION ALIGN=TOP> (заголовок помещается над таблицей) и <CAPTION ALIGN=BOTTOM> (заголовок помещается под таблицей). Если применить <CAPTION ALIGN=LEFT>, заголовок будет начинаться с левого края таблицы; <CAPTION ALIGN=RIGHT> – название заголовка закончится по правому краю; <CAPTION ALIGN=CENTER> – центр заголовка совпадет с центром таблицы.

Как было отмечено выше, каждая строка таблицы начинается с тега <TR> и заканчивается тегом </TR>. Между этими тегами должна быть хотя бы одна ячейка. <TR> может включать следующие атрибуты:

- ALIGN – устанавливает выравнивание текста в ячейках строки. Допустимые значения: ALIGN=LEFT (выравнивание влево), ALIGN=CENTER (выравнивание по центру), ALIGN=RIGHT (выравнивание вправо).
- VALIGN – устанавливает вертикальное выравнивание текста в ячейках строки. Допустимые значения: VALIGN=TOP (выравнивание по верхнему краю), VALIGN=MIDDLE (выравнивание по центру), VALIGN=BOTTOM (выравнивание по нижнему краю).
- BACKGROUND – задает URI фонового рисунка для строки.
- BORDERCOLOR – задает цвет рамки.
- BORDERCOLORDARK – задает цвет "затененной" части рамки.
- BORDERCOLORLIGHT – задает цвет "светлой" части рамки.

Каждая ячейка таблицы начинается с тега <TD> и заканчивается тегом </TD>. Тег <TD> может включать следующие атрибуты:

- NOWRAP – запрещает перенос строк.
- COLSPAN – устанавливает протяженность ячейки по горизонтали. Например, COLSPAN=3 означает, что ячейка простирается на три колонки.
- ROWSPAN – устанавливает протяженность ячейки по вертикали. Например, ROWSPAN=2 означает, что ячейка занимает две строки.
- BORDERCOLOR – задает цвет рамки ячейки.
- BORDERCOLORDARK – задает цвет "затененной" части рамки.
- BORDERCOLORLIGHT – задает цвет "светлой" части рамки.

Желаемая высота и ширина ячейки указываются с помощью параметров, аналогичных параметрам для таблицы:
WIDTH=число | WIDTH=процент%
HEIGTH=число | HEIGTH=процент%

Аналогично цвету фона и фоновому рисунку для страницы в целом можно задать их для ячейки таблицы. Параметры те же.

Если ячейка таблицы пуста, вокруг нее не рисуется рамка. Если ячейка пуста, а рамка нужна, в ячейку можно ввести символьный объект   (non-breaking space – неразрывающий пробел). Ячейка по-прежнему будет пустой, а рамка вокруг нее будет. Для лучшего форматирования таблицы все же полезно создать рисунок с прозрачным фоном самого маленького размера, а затем вставлять в те ячейки, что должны быть пустыми, указывая нужные размеры.

Следует отметить, что любая ячейка таблицы может содержать в себе другую таблицу.

Рассмотрим несколько примеров построения таблиц.

Пример 1. Таблица 3 × 2

A	B	C
D	E	F

```
<TABLE BORDER=2>
    <TR>
        <TD>A</TD> <TD>B</TD> <TD>C</TD>
    </TR>
    <TR>
        <TD>D</TD> <TD>E</TD> <TD>F</TD>
    </TR>
</TABLE>
```

Пример 2. Объединение вертикальных ячеек

Item 1	Item 2	Item 3
Item 4		Item 5

```
<TABLE BORDER>
    <TR>
        <TD>Item 1</TD>
        <TD ROWSPAN=2>Item 2</TD>
        <TD>Item 3</TD>
    </TR>
    <TR>
        <TD>Item 4</TD> <TD>Item 5</TD>
    </TR>
</TABLE>
```

Пример 3. Объединение горизонтальных ячеек

Item 1	Item 2	
Item 3	Item 4	Item 5

```
<TABLE BORDER>
    <TR>
        <TD>Item 1</TD>
        <TD COLSPAN=2>Item 2</TD>
    </TR>
    <TR>
        <TD>Item 3</TD> <TD>Item 4</TD> <TD>Item 5</TD>
    </TR>
</TABLE>
```

Пример 4. Сложная таблица

		Возраст	
		Рост	**Вес**
Пол	**Мужчины**	1.9	84
	Женщины	1.7	63

```
<TABLE BORDER>
      <TR>      <TD><TH ROWSPAN=2></TH>
            <TH COLSPAN=2>Возраст</TH></TD>
      </TR>
      <TR>
<TD><TH>Рост</TH><TH>Вес</TH></TD>
      </TR>
      <TR>      <TH ROWSPAN=2>Пол</TH>
<TH>Мужчины</TH><TD>1.9</TD><TD>84</TD>
      </TR>
      <TR>
<TH>Женщины</TH><TD>1.7</TD><TD>63</TD>
      </TR>
</TABLE>
```

П р и м е р 5 . Использование **CELLPADDING** и **CELLSPACING**

A	B	C
D	E	F

```
<TABLE BORDER CELLPADDING=10 CELLSPACING=0>
      <TR>
            <TD>A</TD> <TD>B</TD> <TD>C</TD>
      </TR>
      <TR>
            <TD>D</TD> <TD>E</TD> <TD>F</TD>
      </TR>
</TABLE>
```

A	B	C
D	E	F

```
<TABLE BORDER CELLPADDING=0 CELLSPACING=10>
      <TR>
            <TD>A</TD> <TD>B</TD> <TD>C</TD>
      </TR>
      <TR>
            <TD>D</TD> <TD>E</TD> <TD>F</TD>
      </TR>
</TABLE>
```

A	B	C
D	E	F

```
<TABLE BORDER CELLPADDING=10 CELLSPACING=10>
      <TR>
            <TD>A</TD> <TD>B</TD> <TD>C</TD>
      </TR>
      <TR>
            <TD>D</TD> <TD>E</TD> <TD>F</TD>
      </TR>
</TABLE>
```

A	B	C
D	E	F

```
<TABLE BORDER=5 CELLPADDING=10 CELLSPACING=10>
      <TR>
```

```
        <TD>A</TD> <TD>B</TD> <TD>C</TD>
    </TR>
    <TR>
        <TD>D</TD> <TD>E</TD> <TD>F</TD>
    </TR>
</TABLE>
```

П р и м е р 6. Вложенные таблицы
Таблица ABCD внутри таблицы 123456

1	2	3
		A B
		C D
4	5	6

```
<TABLE BORDER>
    <TR> <!-- ROW 1, TABLE 1 -->
        <TD>1</TD>
        <TD>2</TD>
        <TD>3
        <TABLE BORDER>
            <TR> <!-- ROW 1, TABLE 2 -->
                <TD>A</TD>
                <TD>B</TD>
            </TR>
            <TR> <!-- ROW 2, TABLE 2 -->
                <TD>C</TD>
                <TD>D</TD>
            </TR>
        </TABLE>
        </TD>
    </TR>
    <TR> <!-- ROW 2, TABLE 1 -->
        <TD>4</TD>
        <TD>5</TD>
        <TD>6</TD>
    </TR>
</TABLE>
```

П р и м е р 7. Задание ширины и высоты таблицы
Ширина таблицы 50 %

ширина=50%	ширина=50%
3	4

```
<TABLE BORDER WIDTH="50%">
<TR><TD>ширина=50%</TD><TD>ширина=50%</TD>
    </TR>
    <TR><TD>3</TD><TD>4</TD>
    </TR>
</TABLE>
```

Высота=15%

Высота=15%	Item 2
3	4

```
<TABLE BORDER WIDTH="50%" HEIGHT="15%">
    <TR><TD>Высота=15%</TD> <TD>Item 2</TD>
    </TR>
    <TR><TD>3</TD><TD>4</TD>
    </TR>
</TABLE>
```

2.10. ФОРМЫ

Форма – это совокупность стандартных HTML-конструкций ввода текстовой и прочей информации и программы-обработчика этой информации, работающей на Web-сервере. Иными словами, пользовательская форма (или HTML-форма) служит для передачи информационных данных серверу.

Формы являются наиболее популярным средством для интерактивного взаимодействия с пользователем. Они могут содержать такие элементы управления, как кнопки, переключатели, поля для ввода текста и т.д. Все эти элементы можно использовать для того, чтобы посетитель страницы мог отправить информацию на сервер. Посетитель Web-страницы вводит в HTML-форму определенные данные, которые обрабатываются программой и отсылаются на Web-сервер. Все эти действия укладываются в три стадии:

1. Ввод пользователем информации.
2. Обработка введенной информации программой, установленной на сервере.
3. Получение результата отправления введенной информации на Web-сервер (открытие нового HTML-документа, переадресация на предыдущую страницу и пр.).

В качестве программы-обработчика чаще всего выступает CGI-сценарий (скрипт, который обычно разрабатывается на языке Perl или C/C++ и который взаимодействует со специальным компонентом Web-сервера – Common Gateway Interface) или программы, написанные на основе таких серверных языков программирования, как PHP, ASP, JAVASCRIPTP и др. В последнее время формы часто используют совместно с языком JavaScript, например, для организации и (или) настройки пользовательского интерфейса на динамических страницах Web.

Формы передают информацию программам-обработчикам в виде пар [имя переменной]=[значение переменной]. Имена переменных следует задавать латинскими буквами. Значения переменных воспринимаются обработчиками как строки, даже если они содержат только цифры.

Форма открывается тегом <FORM> и заканчивается тегом </FORM>. Элемент FORM служит для объединения всех составных частей формы (полей ввода, кнопок, переключателей) в единое целое. HTML-документ может содержать в себе несколько форм, однако формы должны различаться по именам и не должны находиться одна внутри другой. HTML-текст может размещаться внутри форм без ограничений.

Тег <FORM> может содержать три атрибута, один из которых является обязательным. При помощи атрибутов элемента FORM задается программа на сервере, которая будет обрабатывать информацию, введенную в форму, а также способ отправки этой информации. Это следующие атрибуты:

- **ACTION.** Параметр action является единственным обязательным параметром тега <FORM> и предназначен для указания пути на Web-сервере к программе-обработчику данных пользовательской формы:

<FORM action="/cgi-bin/form.cgi">

Путь к программе-обработчику может быть как относительный, так и абсолютный.

Значение этого параметра играет важную роль с точки зрения работоспособности HTML-формы. Если путь или название программы указаны неверно или же указанный файл не является исполняемым на стороне сервера, то обработка введенных пользователем данных может быть нарушена, произведена некорректно или вовсе не быть осуществленной.

Наиболее распространенными типами исполняемых на стороне сервера файлами являются CGI-программы (с расширениями pi, cgi, fcgi), PHP-скрипты (php, php3, phtm, phtml), ASP-приложения (asp), JAVASCRIPTP-обработчики (JavaScriptp) и др.

- **METHOD.** Параметр method используется для определения типа передачи данных на Web-сервер. Возможными значениями параметра являются get (по умолчанию) и post.

При использовании типа get данные пользовательской формы пересылаются в составе адреса запроса браузера: после имени программы-обработчика ставится знак вопроса (?), обозначающий вывод запроса браузера к переменным HTML-формы, а также последовательность переменных и их свойств из самой формы. Последовательность переменных формы разделяется символом амперсанда "&".

П р и м е р адреса запроса браузера с использованием типа передачи данных GET:

http://www.site.ru/cgi-bin/form.cgi?Name=Vasya&Email=vasya@vasya.ru

Из структуры ссылки, образовавшейся в ходе обработки данных формы, понятно, что пользователь ввел свое имя ("Vasya") и адрес электронной почты ("vasya@vasya. ru").

При использовании типа post данные формы не отображаются в адресной строке браузера, а передаются в составе самого запроса программы-обработчика. Таким образом, используя этот тип, мы получим следующую гиперссылку (учитывая те же данные формы, что и в случае с типом get):

http://www.site.ru/cgi-bin/form.cgi

Следует отметить, что пользовательская форма может включать информацию о пользователе, носящую конфиденциальный характер (например, пароли доступа), которая будет отображена в запросе браузера при использовании типа get. В этом случае вся секретная информация, введенная пользователем, будет доступна для просмотра любому пользователю данного компьютера (просмотр истории перехода по Web-сайтам в браузере легко обнаружит ссылку с конфиденциальными данными).

Но в конечном итоге выбор типа передачи данных на Web-сервер определяется конкретной задачей, стоящей перед автором HTML-формы, особенностями сервера и самой программы-обработчика.

- **ENCTYPE.** Параметр ENCTYPE предназначен для определения типа данных при кодировании информации, введенной пользователем, и последующей ее передаче на Web-сервер. Кодирование осуществляется браузером и используется для предотвращения разного рода искажений в процессе передачи на сервер.

Возможными значениями параметра могут быть: application/x-www-form- urlencoded (по умолчанию) и multipart/form-data.

Первое значение используется, если помимо текста необходимо передать на сервер данные иного типа (к примеру, графику или запакованные файлы). Формат записи состоит из указания типа и его подтипа. Тип данных – это определение общего типа данных (текст, графика, архив, программа и т. д.), например, text, image, application. Подтип – это вид данных внутри определенного общего типа (image/gif, text/html).

Значение multipart/form-data используется в редких специфических случаях, например, при необходимости предоставить пользователю возможность загрузки на сервер любого файла со своего локального компьютера.

- **NAME.** Параметр name присваивает HTML-форме уникальное имя, которое используется в программе-обработчике для идентификации пользовательских данных, например:

Внутри тега <FORM> могут размешаться дополнительные теги и параметры, используемые для включения в состав пользовательской формы различных элементов управления. Они-то и определяют визуальное наполнение HTML-формы (тег <FORM>, и его параметры не видимы пользователю и предназначены для указания необходимой информации браузеру и программе-обработчику на сервере).

Тег <INPUT> является наиболее распространенным и способен отобразить широкий набор элементов управления пользовательской формой:

- текстовую строку;
- поле ввода пароля;
- поле выбора локального файла для загрузки на Web-сервер;
- опцию выбора;
- опцию переключения;
- кнопку отправления пользовательских данных;
- графический вариант кнопки отправления пользовательских данных;
- кнопку сброса введенных пользовательских данных;
- скрытые поля.

Тег не является парным и всегда используется совместно с атрибутом TYPE. Параметр TYPE позволяет указывать один из перечисленных элементов формы, каждому из которых соответствует отдельное значение.

Каждый элемент INPUT должен включать атрибут name=[имя], определяющий имя элемента (и, соответственно, имя переменной, которая будет передана обработчику). Имя должно задаваться только латинскими буквами. Большинство элементов INPUT должны включать атрибут value="[значение]", определяющий то значение, которое будет передано обработчику под этим именем.

Основные типы элемента <INPUT>:

- type="text"

Определяет окно для ввода строки текста. Может содержать дополнительные атрибуты size="[число]" (ширина окна ввода в символах) и maxlength="[число]" (максимально допустимая длина вводимой строки в символах).

П р и м е р :

<INPUT type="text" size="20" name="user" value="Иван">

Определяет окно шириной 20 символов для ввода текста. По умолчанию в окне находится текст Иван, который пользователь может редактировать. Отредактированный (или неотредактированный) текст передается обработчику в переменной user.

- type="password"

Определяет окно для ввода пароля. Абсолютно аналогичен типу text, только вместо символов вводимого текста показывает на экране звездочки (*).

П р и м е р :

<INPUT type="password" name="pw" SIZE="20" maxlength="10">

Определяет окно шириной 20 символов для ввода пароля. Максимально допустимая длина пароля – 10 символов. Введенный пароль передается обработчику в переменной pw.

- type="radio"

Определяет исключающий переключатель. Может содержать дополнительный атрибут checked (показывает, что именно этот переключатель при открытии формы будет активным). Если создана целая группа исключающих переключателей с одинаковыми именами, то в форме можно будет выбрать только один из них.

П р и м е р (рис. 12):

<INPUT type="radio" name="modem" value="9600" checked> 9600 бит/с

<INPUT type="radio" name="modem" value="14400"> 14400 бит/с

<INPUT type="radio" name="modem" value="28800"> 28800 бит/с

Определяет группу из трех радиокнопок, подписанных 9600 бит/с, 14400 бит/с и 28800 бит/с. Первоначально помечена первая из кнопок. Если пользователь не отметит другую кнопку, обработчику будет передана переменная modem со значением 9600. Если пользователь отметит другую кнопку, обработчику будет передана переменная modem со значением 14400 или 28800.

- type="checkbox"

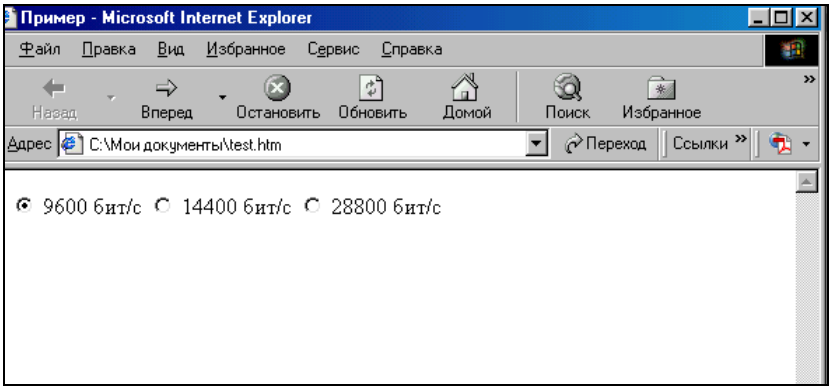

Рис. 12

Создает включатель, в котором можно сделать пометку. Может содержать дополнительный атрибут checked (показывает, что включатель помечен). В отличие от радиокнопок, в группе включателей с одинаковыми именами может быть несколько помеченных включателей.

П р и м е р (рис. 13):

<INPUT type="checkbox" name="comp" value="PC"> Персональные компьютеры
<INPUT type="checkbox" name="comp" value="WS" checked> Рабочие станции
<INPUT type="checkbox" name="comp" value="LAN'> Серверы локальных сетей
<INPUT type="checkbox' name="comp" value="IS' checked> Серверы Интернет

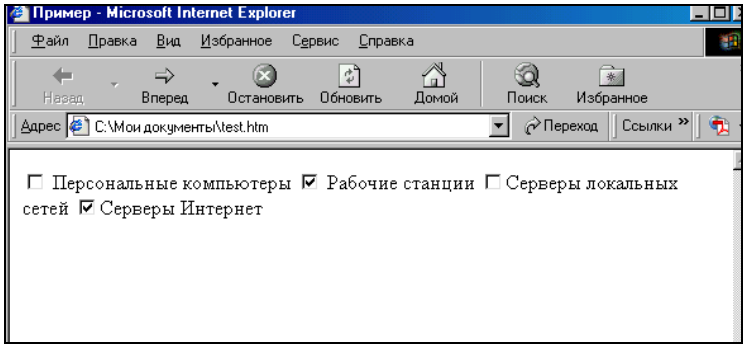

Рис. 13

Определяет группу из четырех включателей. Первоначально помечены второй и четвертый включатель. Если пользователь не произведет изменений, обработчику будут переданы две переменные: comp=WS и comp=IS.

- type="hidden"

Определяет скрытый элемент данных, который не виден пользователю при заполнении формы и передается обработчику без изменений. Такой элемент иногда полезно иметь в форме, которая время от времени подвергается переработке, чтобы обработчик мог знать, с какой версией формы он имеет дело. Другие возможные варианты использования Вы вполне можете придумать сами.

П р и м е р :

<INPUT type="hidden" name="version" value="1.1">

Определяет скрытую переменную version, которая передается обработчику со значением 1.1.

- type="file"

Это элемент выбора файла, расположенного на локальном компьютере пользователя, предназначенного для загрузки на Web-сервер. Из дополнительных параметров действуют только size и maxlength.

Для корректной передачи файла необходима конструкция ENCTYPE="multipart/form-data" и указание типа передачи данных post в теге <FORM>. В противном случае браузер передаст не выбранный файл, а путь к нему на компьютере пользователя.

- type="submit"

Это кнопка отправления пользовательских данных на Web-сервер. При нажатии на нее запускается программа-обработчик, которая анализирует введенные пользователем данные и отправляет результат на сервер.

В принципе, необязательный элемент. Отправление данных может быть выполнено нажатием клавиши <Enter> на клавиатуре пользователя при размещении курсора мыши на любом поле формы. Однако подобный подход не приветствуется, так как отсутствие кнопки подтверждения может запутать неопытного пользователя.

По умолчанию отображается в виде прямоугольной кнопки с надписью "Submit" ("Submit Query" и пр., в зависимости от браузера).

Для элемента отправления данных на сервер также могут использоваться дополнительные (необязательные) параметры – name (имя кнопки отправления, значение которого также будет передано на сервер) и value (присвоение собственного имени кнопке).

Надпись на кнопке можно задать любую, введя атрибут value="Имя кнопки", например:

<INPUT type='submit' value='Поехали!'>

- type="image"

Это графический аналог стандартной кнопки отправления данных формы на Web-сервер. В этом случае автор электронного документа не ограничен средствами HTML и может применить всю свою фантазию на создание оригинальной, привлекательной кнопки передачи данных.

Для данного параметра используются некоторые дополнительные параметры:

src – указание относительного или абсолютного пути на сервере к файлу графического изображения, служащего кнопкой передачи данных формы;

align – указание типа выравнивания текста относительно графической кнопки отправления данных формы;

border – определение толщины рамки кнопки (как правило, значение равно нулю);

alt – указание альтернативного текста/подсказки для кнопки отправления данных.

- type="reset" – это кнопка сброса введенных пользователем данных HTML-формы. При нажатии на нее восстанавливаются все установленные по умолчанию значения полей формы. Элемент не обязателен, однако в ряде случаев весьма полезен. При работе с многочисленными текстовыми строками и опциями выбора пользователь может, допустив ошибку, пожелать заново заполнить форму, тогда ему придется либо перезагружать страницу, либо вручную удалять текст из каждого поля формы. Кнопка сброса в этом случае позволит добиться желаемого при одном нажатии на нее.

По умолчанию названием кнопки сброса является "Reset". Изменить наименование элемента можно с помощью дополнительного параметра value.

П р и м е р :

```
<INPUT TYPE="reset" VALUE="Очистить поля формы">
```

Определяет кнопку **Очистить поля формы**, при нажатии на которую форма возвращается в исходное состояние.

Рассмотрим простейшую форму, которая будет содержать только один элемент – кнопку (рис. 14).

```
<HTML>
<HEAD>
<TITLE>Пример</TITLE>
</HEAD>
<H1>Простейшая форма </H1>
<FORM ACTION="prog6.html"> <!--Это начало формы-->
<INPUT TYPE=submit VALUE="Назад, к главе 6...">
</FORM> <!--Это конец формы-->
</BODY>
</HTML>
```

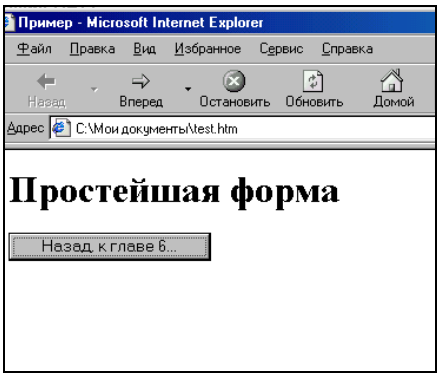

Рис. 14

В форме может быть несколько кнопок типа submit с различными именами и/или значениями. Обработчик, таким образом, может действовать по-разному в зависимости от того, какую именно кнопку submit нажал пользователь.

Помимо элементов INPUT, формы могут содержать элемент SELECT, поля для ввода многострочного текста TEXTAREA и элемент LABLE.

Тег <SELECT> предназначен для компактной группировки большого количества элементов пользовательской формы. К примеру, размещение нескольких десятков элементов CHECKBOX или RADIO займет слишком много места на странице, тогда как группировка данных с помощью тега <SELECT> позволяет заметно сократить размеры, занятые под отображение HTML-формы.

Такой вид формы может быть представлен ниспадающим меню или списком наименований

Элемент SELECT из n элементов выглядит примерно так:

```
<SELECT NAME="[имя]">
<OPTION VALUE="[значение 1]">[текст 1]
<OPTION VALUE="[значение 2]">[текст 2]
...
<OPTION VALUE="[значение n]">[текст n]
</SELECT>
```

и предназначен для создания списка выбора.

Меню начинается с тега SELECT. Тег-тег <SELECT> определяет структуру и вид группировки элементов формы и требует закрывающего тега.

Тег SELECT может включать в себя следующие параметры:

- name – уникальное имя, предназначенное для идентификации программой-обработчиком. Является обязательным параметром, значение которого передается на Web-сервер;
- size – параметр, значение которого определяет число позиций ниспадающего меню, состоящего из списка наименований;
- multiple – параметр, разрешающий выбор нескольких позиций из списка наименований (выбор осуществляется с помощью курсора мыши при нажатой клавише <Ctrl>).

Внутри тега-тега <SELECT> не может находиться никакой информации, а также прочих тегов и их параметров, за исключением тега <OPTION>, который задает свойства для каждой из позиций ниспадающего меню или целого списка наименований.

В теге-теге <OPTION> могут быть размещены два основных параметра:

- value – параметр, значение которого передается программой-обработчиком на Web-сервер. В данном случае будет отправлено значение конкретной выбранной позиции, а не общего значения меню или списка. Следует отметить, что при отсутствии значения параметра value на сервер будет передано содержимое выбранной позиции (т.е. тега-тега <OPTION>);
- selected – параметр отмечает текущую позицию ниспадающего меню или списка наименований как выбранную. Таким образом возможно сделать визуальный акцент на любой по счету позиции меню или списка.

Следует отметить, что тег <OPTION> не требует обязательного присутствия закрывающего тега.

Разберем небольшой п р и м е р , показывающий раскрывающийся список.

```
<HTML>
<HEAD>
</HEAD>
<H2>Работа элемента <FONT COLOR="#FF0000"> SELECT </FONT>.</H2>
<SELECT NAME="selection">
<OPTION VALUE="option1" checked>Вариант 1
<OPTION VALUE="option2">Вариант 2
<OPTION VALUE="option3">Вариант 3
</SELECT>
</BODY>
</HTML>
```

Такой фрагмент определяет меню из трех элементов: Вариант 1, Вариант 2 и Вариант 3. По умолчанию выбран элемент Вариант 1 (рис. 15). Обработчику будет передана переменная selection, значение которой может быть option1 (по умолчанию), option2 или option3.

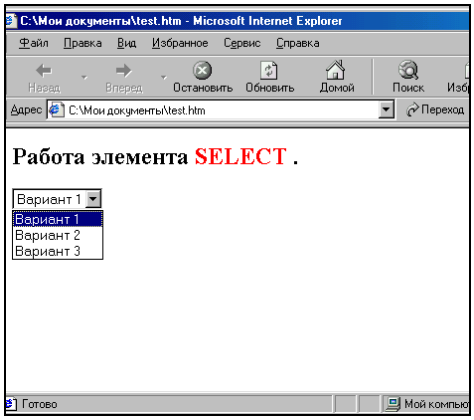

Рис. 15

Еще один маленький п р и м е р показывает нераскрывающийся список (рис. 16):

```
<HTML>
<HEAD>
</HEAD>
<H1>Работа элемента <FONT COLOR="#FF0000"> SELECT </FONT>. </H1>
<SELECT NAME="selection" size=3>
<OPTION VALUE="option1" checked>Вариант 1
<OPTION VALUE="option2">Вариант 2
<OPTION VALUE="option3">Вариант 3
</SELECT>
</BODY>
</HTML>
```

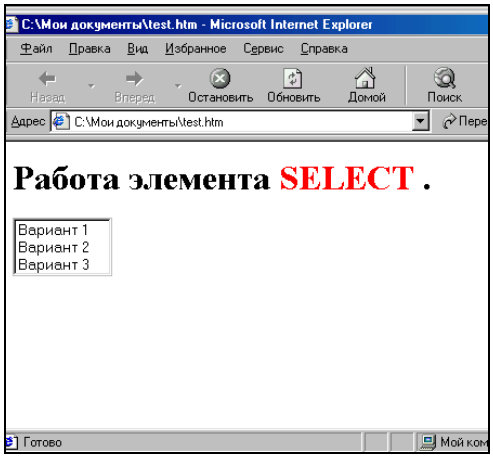

Рис. 16

Тег-тег <TEXTAREA> создает элемент текстового поля заданной ширины и высоты. Указание закрывающего тега обязательно. Внутри тега <TEXTAREA> не может быть никаких других тегов или параметров.

Например:

<TEXTAREA NAME="address" ROWS="5" COLS="50">
А здесь – Ваш адрес...
</TEXTAREA>

Главное отличие элемента текстового поля от элемента текстовой строки заключается в визуальном преимуществе объема вводимой информации.

Атрибут name определяет имя, под которым содержимое окна будет передано обработчику (в примере – address). Атрибут rows устанавливает высоту окна в строках (в примере – 5). Атрибут cols устанавливает ширину окна в символах (в примере – 50). Все атрибуты обязательны.

Текст, размещенный между метками <TEXTAREA> и </TEXTAREA>, представляет собой содержимое окна по умолчанию. Пользователь может его отредактировать или просто стереть.

Важно знать, что русские буквы в окне <TEXTAREA> при передаче обработчику могут быть конвертированы в соответствующие им символьные объекты.

Чтобы обработать форму, необходимо написать программу на одном из скриптовых языков, например, на языке Perl, и положить ее на сервер в виде CGI-скрипта. При нажатии кнопки "Submit" происходит поиск соответствующего скрипта на указанном сервере, который и обрабатывает заполненные поля формы. В нашем примере при нажатии на кнопку "Submit" ничего происходить не будет – ведь это не настоящая форма.

<HTML>
<HEAD>
</HEAD>
<H1>Несколько более сложная форма </H1>
<FORM ACTION="http://206.31.82.215/hp/nc/fd-win.pht" METHOD=post>
<H2>Расскажите немного о себе...</H2>
<P>Указывать подлинные данные совсем не обязательно.
Для целей демонстрации вполне подойдут и вымышленные. </P>
<P>Имя: <INPUT TYPE=text SIZE=40 NAME=fn>
Фамилия: <INPUT TYPE=text SIZE=40 NAME=ln>
Пол: <INPUT TYPE=radio NAME=gender VALUE="male" checked>мужской
<INPUT TYPE=radio NAME=gender VALUE="female">женский
Возраст: <INPUT TYPE=text SIZE=5 NAME=age> лет
<INPUT TYPE=submit VALUE="Запустить обработчик"></P>
</FORM>
</BODY>
</HTML>

Работа этой формы представлена на рис. 17.

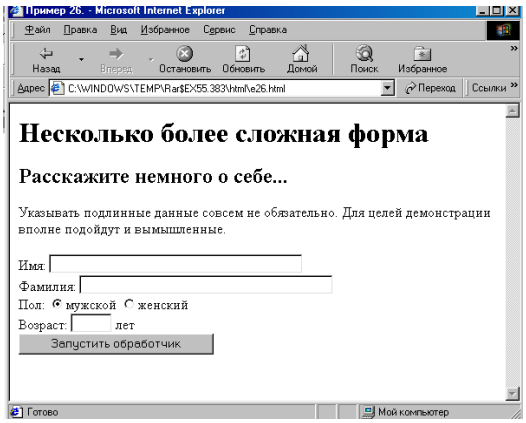

Рис. 17

2.11. ФРЕЙМЫ

Фреймы позволяют разбить окно просмотра браузера на несколько прямоугольных подобластей, располагающихся рядом друг с другом. В каждую из подобластей можно загрузить отдельный HTML-документ, просмотр которого осуществляется независимо от других. Между фреймами, так же, как и между отдельными окнами браузера, при необходимости можно организовать взаимодействие, которое заключается в том, что выбор ссылки в одном из фреймов может привести к загрузке нужного документа в другой фрейм или окно браузера.

Возможность работы с фреймами впервые была реализована в браузере Netscape 2.O. Следующая версия браузера Netscape 3.0 обогатила возможности фреймов, добавив несколько дополнительных параметров к основным тегам описания структуры фреймов. Браузер Microsoft Internet Explorer поддерживает фреймы, начиная с версии 3.0, а также предоставляет уникальную возможность создания плавающих фреймов.

Разработчикам HTML-документов предоставляется довольно богатый выбор форм отображения информации на страницах. Текстовая и графическая информация может быть упорядочена и организована при помощи списков, таблиц или просто с помощью параметров выравнивания, задания горизонтальных линий, разделения на абзацы. Иногда этих возможностей оказывается недостаточно и тогда приходится разбивать окно просмотра браузера на отдельные области или *фреймы* (frames). В ряде русскоязычных описаний языка HTML вместо термина *фреймы* используется термин *кадры*. Частота использования обоих терминов примерно одинакова.

Выбор фреймовой структуры отображения информации на WWW оправдан в следующих случаях:

– при необходимости организовать управление загрузкой документов в одну из подобластей окна просмотра браузера при работе в другой подобласти;

– для расположения в определенном месте окна просмотра информации, которая должна постоянно находиться на экране вне зависимости от содержания других подобластей экрана;

– для представления информации, которую удобно расположить в нескольких смежных подобластях окна, каждая из которых может просматриваться независимо.

Приведенный список не исчерпывает все возможные случаи, где можно применить фреймы, а носит рекомендательный характер.

`Рассмотрим пример фреймсодержащего документа.` На рис. 18 показана одна из HTML-страниц агентства "Финмаркет", специализирующегося на предоставлении информации с финансовых и фондовых рынков России.

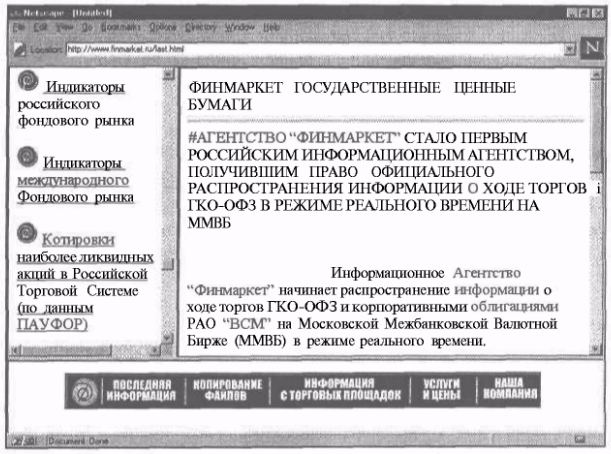

Рис. 18

На этой странице окно браузера разбивается на три фрейма. Нижняя часть окна занимает 20 % высоты всего окна и содержит постоянную информацию, которая в данном случае представляет собой графическое меню, позволяющее в любой момент обратиться к наиболее важным разделам. Этот фрейм не может изменять своих размеров по командам пользователя и не имеет полос прокрутки. Верхняя часть окна (составляющая 80 % высоты) разделена по горизонтали на два фрейма. Ле-

вый фрейм содержит оглавление документов, которые могут быть просмотрены пользователем. Правый фрейм, занимающий большую часть окна просмотра, предназначен для отображения самих документов. При первоначальной загрузке эти два фрейма делят окно браузера по горизонтали в соотношении 15 % на 85 %. Это соотношение может изменяться пользователем при просмотре, что позволяет выбрать оптимальные размеры фреймов с учетом содержимого загруженных документов. Каждый из этих фреймов имеет свою полосу прокрутки, обеспечивающую возможность просмотра всего содержимого фрейма вне зависимости от размера самого фрейма, всего окна браузера и используемых шрифтов. При выборе любой ссылки в левом фрейме соответствующий документ будет загружен в правый фрейм. Такая структура позволяет одновременно видеть на экране и оглавление документов, и содержимое выбранного документа.

Данный пример показывает наиболее типичное использование фреймовых структур, когда один фрейм служит оглавлением документов, а другой используется для загрузки их содержимого. Решение такой задачи без применения фреймов обычно выполняется следующим образом. На одной из страниц располагают оглавление, состоящее из ссылок на другие документы или их отдельные фрагменты. При переходе по такой ссылке оглавление исчезает, а на его место загружается нужный документ, после прочтения которого обычно необходимо вновь вернуться к оглавлению. При использовании фреймов такой возврат становится ненужным, так как оглавление постоянно располагается на части экрана.

Фреймы очень похожи на таблицы – и те и другие осуществляют разбиение окна просмотра браузера на прямоугольные области, в которых располагается некоторая информация. Однако при помощи фреймов можно решить не только задачу форматирования страниц документа, но организовать взаимодействие между ними. Принципиальная разница между фреймами и таблицами состоит в том, что каждому фрейму должен соответствовать отдельный HTML-документ, а содержимое всех ячеек таблицы всегда является частью одного документа.

Если требуется только отформатировать документ, то для этого достаточно ограничиться применением таблиц. Если же необходимо решить более сложные задачи, например, организовать взаимодействие между подобластями окна или создать подобласти, постоянно расположенные на экране, то здесь удобно применить фреймы.

В конечном итоге, выбор структуры документа – табличной или фреймовой – зависит от многих факторов и не может быть однозначно предопределен.

Перейдем теперь к рассмотрению правил записи тегов, используемых для документов с фреймовыми структурами.

Для создания фреймов необходимо иметь те страницы, которые будут загружаться в отдельные фрагменты окна. Кроме этого, необходимо дополнительно создать файл, определяющий структуру документа. HTML-файлы, содержащие набор фреймов, существенно отличаются от обычных HTML-файлов. В частности, они не должны содержать тег <BODY> . Их основная часть определяется тегом <FRAMESET>. Все, что заключено между ним и его закрывающим тегом </FRAMESET>, является набором фреймов. Так как для страниц с фреймами не применяется раздел BODY, то нет возможности задать фоновое изображение и цвет фона для всей страницы в целом. Однако это не мешает в каждый фрейм загружать документы, имеющие свои параметры фона. Внутри тега <FRAMESET> могут содержаться только теги <FRAME> и вложенные теги <FRAMESET>.

Тег <FRAMESET> имеет два атрибута: rows (строки) и cols (столбцы) и записывается в следующем виде:

<FRAMESET rows="список значений" cols="список значений">.

Можно определить значения для rows или cols, или обоих вместе. Необходимо определить, по меньшей мере, два значения хотя бы одного из этих параметров. Если другой параметр опущен, то его значение принимается равным 100 %.

Список значений параметров rows и cols тега <FRAMESET> представляет собой разделенный запятыми список значений, которые могут задаваться в пикселях, в процентах или в относительных единицах. Число строк или столбцов определяется числом значений в соответствующем списке. Например, запись <FRAMESET rows="100,240,140"> определяет набор трех фреймов. Эти значения представляют собой абсолютные значения в пикселях. Другими словами, первый фрейм (первая строка) имеет высоту 100 пикселей, второй – 240 и последний – 140 пикселей.

Задание значений размеров фреймов в пикселях не очень удобно. Здесь не учитывается тот факт, что браузеры запускаются в различных операционных системах и с различными разрешениями дисплеев. В то же время можно определить абсолютные значения размеров для некоторых случаев, например, для отображения небольшого изображения с известными размерами. Лучшим вариантом будет задание значений в процентах или в относительных единицах, например:

<FRAMESET rows="25%,50%,25%">.

В этом примере создаются три фрейма, размещаемые как строки во всю ширину экрана. Верхняя строка займет 25 % от доступной высоты экрана, средняя строка – 50 и нижняя – 25 %. Если сумма заданных процентов не равна 100 %, то значения будут пропорционально отмасштабированы, чтобы в итоге получилось ровно 100 %.

Значения в относительных единицах выглядят следующим образом:

<FRAMESET cols="*,2*,3*">.

Звездочка (*) используется для пропорционального деления пространства. Каждая звездочка представляет собой одну часть целого. Складывая все значения чисел, стоящих у звездочек (если число опущено, то подразумевается единица), получим знаменатель дроби. В этом примере первый столбец займет 1/6 часть от общей ширины окна, второй столбец – 2/6 (или 1/3), а последний – 3/6 (или 1/2).

Числовое значение без каких-либо символов определяет абсолютное число пикселей для строки или колонки. Значение со знаком процента (%) определяет долю от общей ширины (для cols) или высоты (для rows) от окна просмотра, а значение со звездочкой (*) задает пропорциональное распределение оставшегося пространства.

Приведем пример, использующий все три варианта задания значений:

<FRAMESET cols="100,25%,*,2*">.

В этом примере первый столбец будет иметь ширину 100 пикселей. Второй столбец займет 25 % от всей ширины окна просмотра, третий столбец – 1/3 оставшегося пространства и, наконец, последний столбец – 2/3. Абсолютные значения рекомендуется назначать первыми по порядку слева направо. За ними следуют процентные значения от общего размера пространства. В заключение записываются значения, определяющие пропорциональное разбиение оставшегося пространства.

Если вы используете абсолютные значения параметров cols или rows, то задавайте их небольшими, чтобы они могли поместиться в любом окне браузера, и дополняйте их, по крайней мере, одним значением, заданным в процентной или относительной форме, для заполнения оставшегося пространства.

Если используется тег <FRAMESET>, в котором заданы значения и cols, и rows, то будет создана сетка из фреймов. Например:

<FRAMESET rows="*,2*,*" cols ="2*,*">.

Эта строка HTML-кода создает сетку фреймов с тремя строками и двумя столбцами. Первая и последняя строки занимают 1/4 высоты каждая, а средняя строка – половину. Первый столбец занимает 2/3 ширины, а второй – 1/3.

Тег <FRAMESET> </FRAMESET> может быть вложен внутрь другого такого же тега.

Содержимое каждого фрейма задается с помощью тега <FRAME>. Тег <FRAME> имеет шесть параметров: src, name, marginwidth, marginheight, scrolling и noresize.

Приведем запись тега <FRAME> со всеми параметрами:

<FRAME src="url" name="window_name" scrolling=yes marginwidth="value" marginheight="value" noresize>.

На практике в теге <FRAME> редко используются одновременно все параметры. Его основным атрибутом является src, значение которого определяет URL-адрес документа, который будет загружен изначально в данный фрейм. Обычно в качестве такого адреса записывается имя HTML-файла, расположенного в том же самом каталоге, что и основной документ. Тогда строка определения фрейма будет выглядеть, например, так:

<FRAME src="sample.htm">.

Параметр name определяет имя фрейма, которое может использоваться для ссылки к данному фрейму. Обычно ссылка задается из другого фрейма, располагающегося на той же самой странице. Например:

<FRAME src="sample.htm" name= " Frame_l " > .

Такая запись создает фрейм с именем "Frame_l", на который может быть выполнена ссылка. Например:

<A href="other.htm" target="Frame_1">Щелкните здесь для загрузки документа other.htm во фрейм с именем Frame_l</A>.

Обратите внимание на параметр target, который ссылается на имя фрейма. Если для фрейма не задано имя, то будет создан фрейм без имени и не будет возможности использовать ссылки на него из другого фрейма. Имена фреймов должны начинаться с алфавитно-цифрового символа.

Параметры marginwidth и marginheight дают возможность устанавливать ширину полей фрейма. Записывается это следующим образом:

marginwidth ="value", где "value" – абсолютное значение в пикселях. Например:

<FRAME marginheight = "5" marginwidth = "7">.

Данный фрейм имеет поля сверху и снизу по 5 пикселей, а слева и справа – по 7 пикселей. Атрибуты marginwidth и marginheight определяют пространство внутри фрейма, в пределах которого не будет располагаться никакая информация. Минимально допустимое значение этих параметров равно единице.

Для фреймов будут автоматически создаваться и отображаться полосы прокрутки, если содержимое фрейма не помещается полностью в отведенном пространстве. Иногда это нарушает дизайн страницы, поэтому было бы удобно иметь возможность управлять отображением полос прокрутки. Для этих целей используется атрибут scrolling. Формат записи:

<FRAME scrolling ="yes | no | auto">.

Атрибут scrolling может принимать три значения: yes, no или auto. Значение auto действует так же, как и в случае отсутствия параметра scrolling. Значение yes вызывает появление полос прокрутки вне зависимости от необходимости этого, а no – запрещает их появление. Например:

<FRAME scrolling=yes>.

Обычно пользователь может изменять размер фреймов при просмотре страницы. Если установить курсор мыши на рамке фрейма, то курсор примет форму, указывающую на возможность изменения размеров, и позволит выполнить перемещение рамки в нужное место. Это иногда нарушает структуру красиво спроектированных фреймов. Для предотвращения возможности изменения пользователем размера фреймов следует воспользоваться атрибутом noresize:

<FRAME noresize >.

Этот параметр не требует никаких значений. Естественно, когда задан параметр noresize для одного из фреймов, то размер любого из смежных фреймов также не может быть изменен. Иногда, в зависимости от расположения фреймов, использования параметра noresize в одном из фреймов будет достаточно, чтобы предотвратить возможность изменения размеров любого из них на экране.

Необходимо записать столько тегов <FRAME>, сколько отдельных фреймов определено при задании тега <FRAMESET>.

Например:

```
<frameset cols="30%,*">
<frame name="first" src="ExFrame1.html">
<frame name="second" src="ExFrame2.html">
</frameset>
```

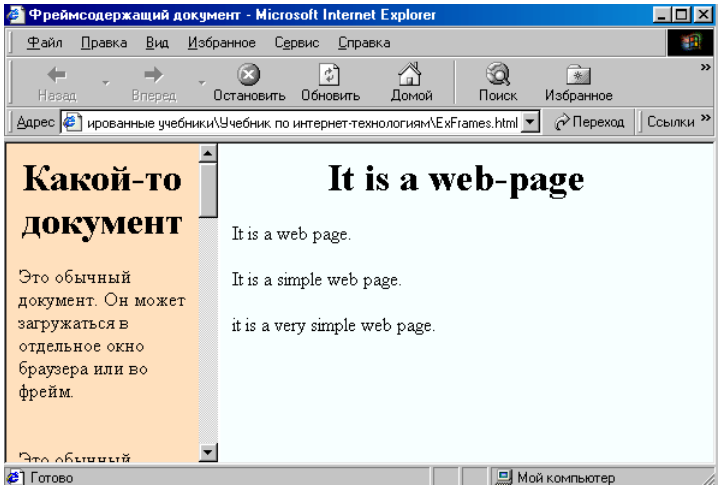

Рис. 19

Результат работы этого кода представлен на рис. 19.

Здесь first и second – имена фреймов, ExFrame1.html и ExFrame2.html – имена документов, загружаемых во фреймы. Атрибут cols означает, что фреймы являются вертикальными (иначе мы использовали бы атрибут rows), согласно присвоенному нами значению: 30 %,* левый фрейм будет занимать 30 % ширины окна браузера, а остальная часть окна будет занята правым фреймом.

Для создания сложных фреймсодержащих документов используются вложенные теги <frameset>...</frameset> или же во фреймы загружаются документы, которые, в свою очередь, также являются фреймсодержащими.

Взаимодействие между фреймами заключается в возможности загрузки документов в выбранный фрейм по командам из другого фрейма. Для этой цели используется атрибут target тега <A>. Данный параметр определяет имя фрейма или окна браузера, в которое будет загружаться документ, на который указывает данная ссылка. По умолчанию при отсутствии атрибута target документ загружается в текущий фрейм (или окно). Задание имени фрейма, в который осуществляется загрузка по умолчанию, очень удобно для тех случаев, когда большое количество ссылок должно направлять документы в определенный фрейм.

Имена фреймов должны начинаться с латинской буквы или цифры. В качестве имени может задаваться имя существующего окна или фрейма, а может указываться новое имя, под которым будет открыто новое окно. Имеются четыре зарезервированных имени, при задании которых выполняются специальные действия. Эти имена начинаются с символа подчеркивания (_): "_blank", "_self", "_parent" и "_top". Любое другое имя, начинающееся с символа "подчеркивание", недопустимо.

- target="_blank" – обеспечивает загрузку документа в новое окно. Это окно не будет иметь имени, а следовательно, в него невозможно будет загрузить другой документ.
- target ="_self" – загрузка документа будет произведена в текущий фрейм (или окно).
- target ="_top" – вызывает загрузку документа в полное окно. Если документ уже располагается в полном окне, то данное значение действует так же, как "_self".
- target ="_parent" – вызывает загрузку документа в область, занимаемую фреймом-родителем текущего фрейма. При отсутствии фрейма-родителя данное значение параметра действует так же, как "_top".

Приведем примеры взаимодействия между фреймами и отдельными окнами браузера. Рассмотрим следующий HTML-код:

```
<HTML>
<HEAD>
<TITLE>Использование фреймов</ TITLE>
</HEAD>
<FRAMESET cols=2 *,*,*>
<FRAME src=frame__a.htm name="A">
<FRAME src=empty.htm name="B">
<FRAME src=empty.htm name="C">
</FRAMESET>
</HTML>
```

В этом HTML-документе дается описание структуры, состоящей из трех фреймов с именами "A", "B" и "C". Имена фреймов потребуются в дальнейшем для организации ссылок между фреймами. Заметим, что на фрейм с именем "A" в данном примере ссылок не будет, поэтому он мог быть оставлен без имени вообще. При загрузке приведенного выше документа в браузер во фреймах будет отображена информация, содержащаяся в файлах, определяемых параметром src. Во фрейм "A" попадет содержимое файла frame_a.htm, а остальные два фрейма получат данные из файла empty.htm, который не имеет отображаемых данных.

Приведем текст файла с именем frame_a.htm:

```
<HTML>
<HEAD>
< TITLE >Документ для фрейма A</TITLE>
```

```
</HEAD>
<BODY>
<A href="test.htm" target="B">1. Загрузка документа во фрейм B</A>
<A href ="test .htm" target ="C">2. Загрузка документа во фрейм C</A>
<A href="test.htm" target="D">3. Загрузка документа в окно с именем D</A>
<A href="test.htm" target="_blank">4. Загрузка документа в новое окно </A>
<A href="test.htm"  target="_top">5. Загрузка документа в полное окно</A>
<A href="test.htm"   target="_self">6. Загрузка документа в текущий фрейм </A>
</BODY>
</HTML>
```

Этот документ является полным HTML-документом, имеющим разделы <HEAD> и <BODY> и, в свою очередь, ссылки на файл с именем test.htm, располагающийся в том же самом каталоге, что и файл frame_a.htm.

Файл frame_a.htm, содержимое которого загрузилось во фрейм "A", имеет шесть ссылок на один и тот же файл test.htm с различным значением атрибута target.

Рассмотрим действия, которые будут происходить при реализации этих ссылок. Первая ссылка со значением target="B" будет загружать файл test.htm во фрейм с именем "B".

Вторая ссылка выполнит те же действия для фрейма "C".

Третья ссылка со значением target="D" приведет к образованию нового окна браузера с именем "D" и загрузке в него файла test.htm. Заметим, что форма записи этой ссылки ничем не отличается от первых двух. Различие состоит в том, что в первых двух случаях ссылки были даны на существующие фреймы, имена которых были определены в файле со структурой фреймов, а в данном случае ссылка дана на несуществующий объект. Если данная ссылка будет выполнена хотя бы один раз, то окно с именем "D" будет образовано и повторный переход по ссылке лишь перезагрузит данные в существующее теперь окно "D". Конечно, пользователь может в любой момент его закрыть и вновь образовать выбором данной ссылки.

Четвертая ссылка со значением target="_blank" создаст новое окно без имени и загрузит туда требуемый документ. Любое повторение данной ссылки будет открывать еще одно окно браузера.

Пятая ссылка со значением target="_top" загрузит документ в полное окно вместо всей фреймовой структуры. При таком значении атрибута target новое окно не образуется. Возврат к фреймовой структуре возможен нажатием кнопки **Back.**

Последняя ссылка со значением target="_seif" загрузит документ во фрейм "A" на место документа со ссылками. В данном случае результат эквивалентен выполнению ссылки без атрибута target.

Имена фреймов или окон браузера не следует путать с названиями загружаемых документов. Имена фреймов при просмотре нигде не видны, они требуются только для организации взаимодействия и поэтому скрыты от пользователя. Увидеть их можно только при просмотре исходного текста HTML-файлов.

Одним из наиболее часто встречающихся вариантов применения фреймов, который уже упоминался в данной главе, является случай двух фреймов, один из которых содержит список ссылок, а в другой загружаются сами документы.

Попробуем расширить постановку задачи. Пусть необходимо отображать на экране содержимое достаточно большого документа, состоящего из глав, разделенных на разделы. Типичным примером служит техническая литература по какой-либо тематике. Опишем желаемое представление такого документа на экране. Разобьем экран на три фрейма, в одном из которых будет располагаться список глав книги, во втором – перечень разделов выбранной главы, а в третьем – текст выбранного раздела. При выборе ссылки во втором фрейме должно меняться содержимое третьего фрейма. Реализация этого требования тривиальна. При выборе ссылки в первом фрейме должно одновременно изменяться содержимое как второго, так и третьего фрейма. На первый взгляд реализация этой задачи на языке HTML невозможна (без применения программирования на языке JavaScript или др.), так как при выполнении ссылки загружается только один документ, а не два или более. Тем не менее, решение данной задачи вполне возможно.

Покажем возможную схему решения такой задачи на простом примере. Пусть требуется отобразить на экране три фрейма и загрузить в них некоторые документы. Поставим задачу создать в каждом из этих фреймов ссылки, реализация которых, например, меняет местами содержимое двух фреймов. Пусть первый фрейм занимает 50 % ширины окна и 100 % высоты и располагается с левой стороны окна. Правая половина окна делится по горизонтали также пополам и содержит два других фрейма. Такая структура описывается следующим кодом:

```
<HTML>
<HEAD>
<TITLE>Пример взаимодействия между фреймами</TITLE>
</HEAD>
<FRAMESET cols="*,*">
<FRAME  src="left.htm">
<FRAMESET  rows="*,*">
<FRAME  src="l.htm">
<FRAME  src="2.htm">
</FRAMESET> </FRAMESET>
```

С помощью данного HTML-кода будет создана требуемая структура, однако решение поставленной задачи невозможно. Необходимо вынести вложенную структуру <FRAMESET> в отдельный файл, а в данном HTML-коде описать фрейм, ссылающийся на созданный файл. Тогда текст исходного документа будет иметь вид:

```
<HTML>
<HEAD>
<TITLE>Пример взаимодействия между фреймами</TITLE>
```

```
</HEAD>
<FRAMESET cols="*,*">
<FRAME src="left.htm">
<FRAME src="l_2.htm" name="Two_Frames">
</FRAMESET>
</HTML>
```

Созданный файл с вложенной структурой <FRAMESET> имеет имя l_2.htm и содержит следующий код:

```
<HTML>
<HEAD>
<TITLE>l-2</TITLE>
</HEAD>
<FRAMESET rows="*,*">
<FRAME src="l.htm">
< FRAME src="2.htm">
</FRAMESET>
</HTML>
```

На первый взгляд совершенно ничего не изменилось. В обоих случаях имеются три фрейма, в которые загружаются документы left.htm, l.htm и 2.htm, соответственно. Однако при взаимодействии фреймов различие проявится. Если в первом случае ни у одного из фреймов нет фрейма-родителя, то во втором случае для двух фреймов родительским будет фрейм с именем "Two_Frames". Поэтому если в любом из двух фреймов применить ссылку со значением параметра target, равным "_parent", то результат будет различным для первого и второго случая. Для первого случая реализация такой ссылки приведет к загрузке документа в полное окно с замещением существующей структуры фреймов. Здесь проявляется свойство значения "_parent", которое при отсутствии фрейма-родителя действует как "_top". Во втором случае будет замещен фрейм с именем "Two_Frames", который занимает правую половину экрана и по существу состоит из двух фреймов.

Второй случай формально отличается от первого также наличием фрейма с именем "Two_Frames", к которому могут быть обращены ссылки. Как раз эта особенность и позволит нам решить поставленную задачу.

Приведем содержимое файла left.htm, который изначально загружается в первый из рассматриваемых фреймов:

```
<HEAD>
<TITLE>Левый фрейм</TITLE>
</HEAD>
<BODY>
```

Реализация любой ссылки во всех трех фреймах приводит к перезагрузке документов в двух фреймах, расположенных в правой части окна.

```
<A href=" l_2.htm" target="Two_Frames"> Вариант 1-2
<P>
<A href="2_l.htm" target="Two_Frames">Вариант 2-1</A>
</BODY> </HTML>
```

В этом документе имеются ссылки на файлы l_2.htm и 2_l.htm. Текст первого был дан выше, а второго имеет следующий вид:

```
<HTML>
<HEAD>
<TITLE>2-1</TITLE>
</HEAD>
<FRAMESET rows="*,*">
<FRAME  src="2.htm">
<FRAME  src="l.htm">
</FRAMESET>
```

Заметим, что тексты файлов l_2.htm и 2_l.htm отличаются только порядком ссылок на файлы l.htm и 2.htm.

Рассмотрим теперь построение документа, загруженного в левый фрейм. В нем имеются две ссылки с параметром target="Two_Frames". Реализация любой из этих ссылок создает на месте расположения фрейма "Two_Frames" (это правая половина экрана) два фрейма с загрузкой документов l.htm и 2.htm в том или ином порядке. Таким образом при выборе варианта 1-2 в верхний правый фрейм загружается документ l.htm, а в нижний правый – 2.htm. При выборе варианта 2-1 порядок документов меняется. В итоге поочередный выбор вариантов создает впечатление того, что документы в двух фреймах меняются местами. Именно такого эффекта мы и стремились достичь.

Содержимое документов l.htm и 2.htm для описанного примера не имеет значения. Тем не менее, для примера, вместо тривиальных документов создадим документы со ссылками, реализующими те же действия.

Текст файла l.htm:

```
<HTML>
<HEAD>
<TITLE>Документ 1</TITLE>
</HEAD>
<BODY>
<H2>Документ 1</H2>
<A href=" 1_2.htm" target="_parent">Вариант 1-2
```

<A href="2_l.htm" target="_parent:">Вариант 2-1</A> </BODY>
</HTML>

Файл 2.htm отличается от 1.htm только заголовком.

Здесь имеются две ссылки со значением target="_parent", которые обращены к родительскому фрейму. Эти ссылки могли бы быть записаны и с явным указанием имени фрейма-родителя, т.е. target="Two_Frames", однако использование неявного указания имени обычно более удобно. Например, если из левого фрейма (документ left.htm) исключить ссылки, то можно было бы опустить имя фрейма "Two_Frames", заданное при описании основной фреймовой структуры. При этом был бы создан фрейм без имени, но ссылки из документов l.htm и 2.htm со значением target="_parent" по-прежнему работали бы правильно.

При работе с фреймами возникает вопрос о принципиальной разнице между организацией фреймовой структуры окна браузера и созданием нескольких окон. На первый взгляд может показаться, что вполне можно было бы обойтись возможностями создания нескольких окон, поскольку работа с окнами и фреймами очень похожа. Каждый фрейм требует загрузки отдельного документа, имеет возможность независимой прокрутки содержимого и может изменяться по командам из других фреймов. Эти свойства фреймов аналогичны свойствам окон браузера. При табличной организации данных добиться такой свободы действий невозможно.

Однако между фреймами и окнами есть существенная разница. При фреймовой организации деление области просмотра на фреймы выполняет сам HTML-документ, указывая размеры и их расположение. Пользователь при просмотре может изменить размеры фреймов, если это не запрещено в описании их структуры. Расположение окон определено общими правилами работы с системой Windows – пользователь может распахнуть любое окно на весь экран, свернуть его в пиктограмму или произвольным образом задать размеры и расположение. Окна, в отличие от фреймов, могут перекрываться. Такое богатство выбора имеет свою оборотную сторону – необходимо каждый раз вручную располагать окна на экране и изменять их размеры для достижения оптимального варианта просмотра. В случае фреймов оптимальный вариант соотношения размеров обычно задается разработчиком в описании фреймовой структуры и часто не нуждается в изменении.

2.12. КАРТЫ-ИЗОБРАЖЕНИЯ

В последнее время многие Web-страницы для организации ссылок используют так называемые *карты-изображения*. Реализация этой возможности предусмотрена языком HTML и позволяет привязывать гипертекстовые ссылки к различным областям изображения. Такой подход нагляднее, чем применение обыкновенных текстовых связей, поскольку пользователь может не читать словесное описание связи, а сразу понять ее смысл по графическому образу.

Примером использования карт-изображений может служить известный поисковый сервер Yahoo! (рис. 20). Самая верхняя часть изображения, приведенного на рисунке, содержит четыре кнопки, между которыми написано слово "Yahoo!". Курсор на рисунке показывает на первую из этих кнопок, причем форма курсора дает понять, что последний указывает на ссылку, адрес которой виден в строке статуса браузера. Ссылки, реализующиеся по этим кнопкам, и сделаны по технологии карт-изображений.

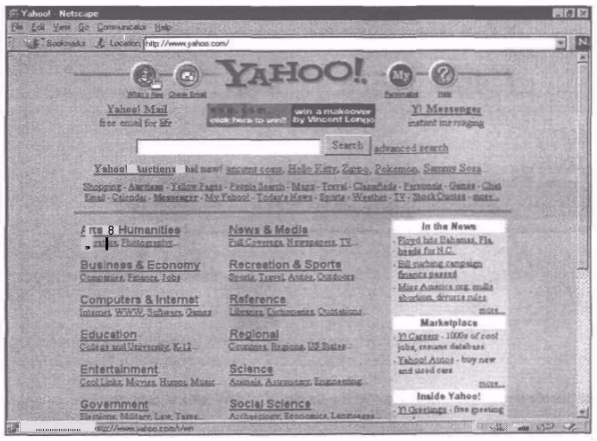

Рис. 20

Однако не следует считать, что карты-изображения должны использоваться всюду, где требуется организовать переходы по ссылкам. Нужно обдумать, имеет ли смысл применение карт-изображений в том или ином случае, взвесив все "за" и "против".

Карты-изображения предоставляют пользователям дружественный интерфейс для перехода на другие Web-страницы. Чтобы выполнить переход по такой ссылке, следует просто выбрать нужное место на изображении и щелкнуть мышью. Наличие такого развитого графического интерфейса является одним из значительных преимуществ Web-страниц по сравнению с другими ресурсами Интернета. Вместо текстовых меню пользователи получают наглядное графическое представление информации.

Карта-изображение внешне выглядит как обычное встроенное изображение, но при выборе с помощью курсора мыши той или иной области на этом изображении выполняется переход на другие страницы. Обычно на изображении указывается, где следует сделать щелчок, чтобы перейти на ту или иную страницу. Существует несколько путей указания границ областей, реализующих различные ссылки. Часто используется рамка или какой-либо иной разделитель.

Imagemap, Image Map, Area Map, Clickable Map, Sensitive Map – все эти англоязычные термины используются в справочной литературе для обозначения одной и той же возможности – использования встроенного в HTML-документ изображения, для которого определены "горячие" (или активные) точки или области, имеющие ссылки на различные URL-адреса. Будем описывать эту возможность словосочетанием "карта-изображение", подразумевая под этим совокупность нескольких компонентов, обеспечивающих реализацию данной концепции. Основными компонентами являются: само изображение, которое будем называть опорным для данной карты-изображения, описание конфигурации активных областей, а также соответствующее программное обеспечение.

Карта-изображение фактически представляет собой обычное встроенное графическое изображение на Web-странице. Эти изображения могут иметь любой допустимый формат (GIF или JPG). При этом в формате GIF может использоваться прозрачный цвет, а также режим чередования строк. Для того чтобы изображение могло использоваться в качестве опорного для карты-изображения, формально не накладывается никаких дополнительных ограничений.

Конфигурация карты-изображения записывается в виде обычного текста, который в зависимости от используемого формата может быть сохранен в отдельном файле или являться частью HTML-документа. Описание конфигурации содержит координаты для каждой из активных областей изображения, а также URL-адреса, связанные с каждой из этих областей. Активные области могут иметь форму прямоугольников, кругов и многоугольников. Допускается любая комбинация этих фигур. Также может задаваться одно значение URL-адреса, определенное для случая, когда пользователь выполняет щелчок в пределах изображения, но вне любой из заданных активных областей.

Концепция карты-изображения на Web-страницах может быть реализована в двух различных вариантах – серверный вариант (server-side imagemap) и клиентский вариант (client-side imagemap). Последнее название часто используют в виде аббревиатуры CSIM. Исторически первым появился и получил распространение серверный вариант карт-изображений. Серверный вариант может быть реализован в двух различных форматах, которые получили свое наименование по названиям организаций-разработчиков – NCSA и CERN.

В последнее время все большее развитие получает клиентский вариант. Данный вариант имеет свои неоспоримые преимущества и становится все более популярным.

В использовании карт-изображений есть как положительные, так и отрицательные моменты. Большинство из них носит эстетический характер, но некоторые имеют и технические аспекты. Для создания хороших Web-страниц важно понимать преимущества и недостатки карт-изображений.

Карты-изображения наиболее удобно использовать в следующих ситуациях:

– для представления пространственных связей, например географических координат, которые было бы трудно задать отдельными кнопками или текстом;

– в качестве меню верхнего уровня, появляющегося на каждой странице. Наличие такого меню предоставляет возможность перехода в интересующий раздел сервера с любой страницы и в любой момент. Создание общего графического меню позволит сократить время разработки HTML-документов, поскольку будет использоваться один и тот же файл описания ссылок. Вместо того, чтобы на каждой странице устанавливать связи с различными частями начальной страницы, достаточно сослаться на общее меню. Такое меню также облегчит навигацию для пользователя.

К недостаткам карт-изображений можно отнести следующие:

– если не предусмотрено альтернативное текстовое меню, то не остается никаких средств навигации для пользователей, которые не могут загрузить графику или отключили ее загрузку;

– картам-изображениям свойственны общие недостатки, присущие использованию изображений на Web-страницах, а именно, значительное увеличение времени загрузки по сравнению с чисто текстовыми документами;

– неудачно спроектированные изображения могут внести путаницу. Иногда бывает трудно определить области, являющиеся активными на изображении. Особенно это трудно сделать в серверном варианте. При реализации клиентского варианта ситуация упрощается, так как есть возможность перемещать мышь в пределах изображения и следить за появляющимися адресами ссылок в нижней части окна браузера;

– при использовании карт-изображений браузер не имеет возможности отмечать другим цветом уже пройденные ссылки так, как это делается для текстовых ссылок.

В данном пособии рассмотрим особенности разработки клиентского варианта карты изображений.

Клиентский вариант карты-изображения позволяет разместить всю информацию о конфигурации карты в HTML-файле, в который встроено изображение. В случае же применения серверного варианта браузер посылает запрос на сервер для получения адреса выбранной ссылки и ждет ответа с требуемой информацией. Это может потребовать дополнительных затрат времени на ожидание. При клиентском варианте число обращений к серверу уменьшается, и увеличивается скорость доступа к информации. В этом варианте также для редактирования конфигурации карты нет необходимости обращения к серверу, поэтому вся работа по созданию карты-изображения может быть выполнена локально, одновременно с редактированием HTML-файла.

Для указания того, что встроенное изображение является опорным для карты, используется атрибут usemap тега <IMG>. Значением атрибута usemap является ссылка на описание конфигурации карты.

Например:

<IMG src=logo.gif usemap=#logo>

В этом примере изображение, хранящееся в файле с именем logo.gif, является опорным для карты-изображения, реализуемой в клиентском варианте.

Описание конфигурации активных областей должно располагаться в том же файле, что и данная строка HTML-кода, и иметь для данного примера имя logo.

Для описания конфигурации областей карты-изображения используется специальный тег <MAP>, единственным атрибутом которого является name. Значение атрибута name определяет имя, которое должно соответствовать имени в usemap.

Тег <MAP> требует закрывающего тега </MAP>. Внутри этой пары тегов должны располагаться описания активных областей карты, для чего используется специальный тег <AREA>.

Каждый отдельный тег <AREA> задает одну активную область. Завершающий тег не требуется. Активные области могут перекрываться. В случае, если некоторая точка относится одновременно к нескольким активным областям, то будет реализована та ссылка, описание которой располагается первым в списке областей.

Атрибутами тега <AREA> являются shape, coords, href, nohref, target, и alt. Рассмотрим назначение этих атрибутов.

Атрибут shape определяет форму активной области. Допустимыми значениями являются rect, circle, poly, default. Эти значения задают области в виде прямоугольника, круга, многоугольника. Последнее значение – default – определяет все точки области. Если атрибут shape опущен, то по умолчанию предполагается значение rect, т.е. область в виде прямоугольника.

Не следует путать область типа default, которая описывает все точки изображения, и значение атрибута shape по умолчанию, которым является rect.

Для клиентского варианта область типа default определяет вообще все точки изображения. Поэтому в данном случае описание области default должно располагаться последним в списке активных областей. Если, например, описание области default поставить первым, то всегда для клиентского варианта будет реализовываться ссылка, определяемая данной областью, а все остальные ссылки будут игнорироваться.

Не все браузеры поддерживают тип области default. В частности, Microsoft Internet Explorer вообще не разрешает использовать данный тип области. Поэтому вместо области типа default можно рекомендовать задание прямоугольной области с размерами, равными размерам всего изображения. Естественно, что такая область должна описываться последней.

Атрибут coords задает координаты отдельной активной области. Значением атрибута является список координат точек, определяющих активную область, разделенных запятыми. Координаты записываются в виде целых неотрицательных чисел. Начало координат располагается в верхнем левом углу изображения, которому соответствует значение 0,0. Первое число определяет координату по горизонтали, второе – по вертикали. Список координат зависит от типа области.

Для области типа rect задаются координаты верхнего левого и правого нижнего углов прямоугольника.

Для области типа circle задаются три числа – координаты центра круга и радиус.

Для области типа poly задаются координаты вершин многоугольника в нужном порядке. Последняя точка в списке координат не обязательно должна совпадать с первой. Если они не совпадают, то при интерпретации данных для этой формы области браузер автоматически соединит последнюю точку с первой. Количественные ограничения на число вершин довольно велики и покрывают практически все мыслимые потребности. По крайней мере многоугольник, имеющий 100 вершин, уверенно обрабатывается всеми ведущими браузерами. Многоугольник вполне может быть невыпуклым.

Область типа default не требует задания координат.

Атрибуты href и nohref являются взаимоисключающими. Если не задан ни один из этих атрибутов, то считается, что для данной области не имеется ссылки. То же самое явно определяет атрибут nohref, не требующий значения. Атрибут href определяет адрес ссылки, который может записываться в абсолютной или относительной форме. Правила записи полностью совпадают с правилами записи ссылок в теге <A>.

Атрибут nohref полезно использовать для исключения части активной области. Пусть, например, необходимо создать активную область в виде кольца. Такой тип области не предусмотрен в списке возможных областей, однако он может быть реализован путем задания двух круговых областей. Для этого сначала следует задать область меньшего радиуса и указать в качестве атрибута nohref. Далее нужно задать область большего радиуса с центром в той же точке и указать нужную ссылку. Тогда область внутри кольца, определенная двумя окружностями различного радиуса, будет иметь необходимую ссылку. Использование подхода, основанного на взаимном перекрытии областей, позволит строить области весьма разнообразной формы.

Атрибут target употребляется при работе с фреймами. Его назначение – указать имя фрейма, в который будет размешен документ, загружаемый по данной ссылке.

Атрибут alt позволяет записать альтернативный текст для каждой из активных областей изображения. По существу этот текст будет играть лишь роль комментария для создателя документа. Если альтернативный текст, записанный для всего изображения (в теге <IMG>), служит для выдачи его на экран при работе с отключенной загрузкой изображений, то альтернативный текст для активных областей никогда на экране не появится.

Пример конфигурации простейшей карты изображений, основой для которой служит изображение, хранящееся в файле logo.gif, а области – различных типов:

```
<IMG src=logo.gif usemap=#logo>
<MAP name="logo">
<AREA shape=rect coords="33,60,191,246" href="r.htm" alt="Прямоугольная область">
<AREA shape=circle coords="366,147,109"  href="c.htm" alt="Круговая область">
<AREA shape=poly coords="534,62,699, 62, 698, 236, 626, 261, 534, 235"
href="p.htm" alt="Многоугольник">
<AREA shape=default href="default.htm">
</MAP>
```

Этот фрагмент кода размещается в HTML-файле. Часто все описания карт-изображений одного документа сводятся вместе и размещаются в начале раздела <BODY> документа.

Применение карт-изображений стало общепринятым, однако не следует забывать о том, что еще не все пользователи Web могут использовать графику или же хотят работать в режиме отключения загрузки изображений для уменьшения времени передачи файлов. Поэтому нужно предусмотреть для них какие-либо другие, альтернативные средства навигации на странице. В противном случае пользователи вообще не смогут обнаружить на странице и, соответственно, реализовать те ссылки, которые определены только картой-изображением.

В качестве альтернативного варианта можно создать отдельный раздел с текстовым описанием ссылок и соответствующих URL-адресов. Можно также создать ссылку на текстовое меню, которое имеет те же самые связи, что и карта-изображение. При любом подходе, какой бы ни был избран, нужно убедиться, что для текстового режима браузера доступны все ссылки.

2.13. ЗВУК

2.13.1. Звуковые форматы

Существует огромное количество звуковых форматов – распространенных и не особо встречающихся, применяемых в различных операционных системах и для различных нужд.

Нас интересуют те форматы, файлы которых позволительно встраивать в HTML-документы и воспроизводить непосредственно в браузере пользователя. К ним можно отнести:

- WAV – формат звуковых файлов операционной системы Microsoft Windows;
- MIDI – Musical Instrument Digital Interface (формат электронных инструментов);
- AU – формат Sun/NeXT;
- AIFF – формат платформы Macintosh;
- RealAudio – формат передачи звуковых файлов в режиме реального времени.

Также в последнее время большую популярность получил формат MP3 (формат сжатия аудиосигнала с потерей качества, лежащей за пределами возможностей человеческого слуха). Однако с помощью стандартных средств HTML файлы MP3 встроить в структуру электронного документа нельзя.

2.13.2. Встраивание звуковых файлов в HTML-документ

Для встраивания практически любых звуковых файлов в HTML-документ можно использовать два пути:

- создание гиперссылки на музыкальный файл и размещение ее на странице;
- применение специального тега вставки музыкальных файлов <EMBED>.

Правила записи гиперссылки на файлы музыкальных форматов абсолютно идентичны записи ссылок на HTML-документы или графику:

<A href="music. wav">Музыкальный файл (30 Кб) </A>

При нажатии на такую ссылку на компьютере пользователя запускается установленный по умолчанию проигрыватель звуковых файлов.

Следует обратить особое внимание на указание размера музыкального файла, стоящее в ссылке сразу после названия. Дело в том, что некоторые звуковые данные (песни, фрагменты мелодий, фонограммы и пр.) могут иметь весьма большой размер. И прежде чем нажать на подобную ссылку, пользователь, учитывая скорость соединения с Интернетом и технические возможности своего компьютера, должен иметь хотя бы примерное представление о том, сколько времени займет открытие файла.

Тег <EMBED> имеет ряд параметров, которые позволяют управлять воспроизведением музыкального файла непосредственно в окне браузера. Рассмотрим пример встраивания звукового файла с помощью тега <EMBED> (рис. 21).

```
<HTML> <HEAD>
<TITLE>Пример встраивания звукового файла</TITLE>
</HEAD>
<BODY>
<EMBED src="melody.wav" width="287" height="213" autostart="true">
</BODY>
</HTML>
```

В этом случае браузер отобразит встроенную панель воспроизведения указанного музыкального файла ("melody.wav")

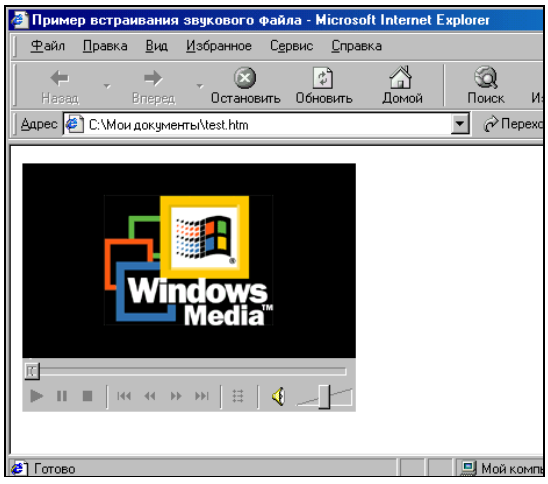

Рис. 21

В теге <EMBED> используются следующие атрибуты:

- src – указание пути к воспроизводимому звуковому файлу форматов AU, AIFF, WAV и MIDI. Является обязательным параметром;
- width – ширина панели воспроизведения (обязательный параметр);
- height – высота панели воспроизведения (обязательный параметр);
- autostart – автоматическое начало воспроизведения. Для автоматического начала воспроизведения используется значение true. Значение по умолчанию – false;
- autoload – разрешение или запрет автоматической загрузки файла. Значение по умолчанию – true. Для запрета автоматической загрузки файла используется значение false;
- align – тип выравнивания панели воспроизведения относительно текста на странице. Возможные значения: left, right, center, top и baseline. Значение по умолчанию – baseline;
- volume – установка громкости воспроизведения (определяется в процентах);
- starttime – время начала воспроизведения в минутах и секундах (формат записи: "mm:ss"; по умолчанию – 00.00);
- endtime – время окончания воспроизведения в минутах и секундах (формат записи: "mm: ss"; по умолчанию время окончания звучания совпадает с концом звукового файла);
- controls – указание элементов управления на панели воспроизведения (значения console, smallconsole, playbutton, pausebatton, stopbutton, volumelever). Значение по умолчанию – console. Указанные значения обозначают следующее: console – отображение полного набора элементов управления на панели воспроизведения; smallconsole – компактный набор элементов управления на панели воспроизведения; playbutton – отображение только кнопки воспроизведения; pausebatton – отображение только кнопки паузы; stopbutton – отображение только кнопки остановки воспроизведения (звуковой файл при этом будет выгружен); volumelever – отображение только регулятора громкости.

Особое внимание при встраивании музыкальных файлов в HTML-документы следует обратить на совместимость перечисленных параметров тега <EMBED> с различными браузерами.

Дело в том, что браузер Internet Explorer использует для воспроизведения встроенных звуковых файлов модуль Active-Movie (проигрыватель Windows Media), а Netscape – LiveAudio. Каждый из этих модулей имеет ряд своих преимуществ, однако результат встраивания музыкального файла визуально может сильно отличаться при просмотре в этих двух браузерах.

2.13.3. Технология RealAudio

Формат RealAudio был разработан компанией RealNetworks в период развития аудио- и видеоконференций в режиме реального времени. Новый формат положил начало возможности передавать звуковые файлы в Интернете в реальном времени.

Технология RealAudio состоит из трех программных компонентов:

- кодировщик – конвертирование звуковых файлов в формат RealAudio;
- сервер – передача файлов формата RealAudio по сети;
- проигрыватель – воспроизведение музыкальных файлов в формате RealAudio.

Для прослушивания RealAudio-файлов достаточно иметь установленный проигрыватель. Для распространения собственных файлов придется обзавестись и программой-кодировщиком, и сервером.

Для встраивания звукового файла в формате RealAudio на HTML-страницу используются также два варианта – посредством ссылки на источник RealAudio-данных и с помощью уже известного тега <EMBED>.

Ссылка на файл RealAudio выглядит так:

pnm://audio.real.com/melody.ra,

где

pnm: // – это указание сервера RealAudio;

audio.real.com – адрес сервера RealAudio;

melody. ra – название звукового файла RealAudio (расширение ra).

Совокупность ссылок такого вида обычно размешают в обыкновенном текстовом файле, содержащем информацию о звуковых данных RealAudio, которому присваивают расширение ram (RealAudio Metafile).

Таким образом позволительно указание обоих расширений (и ra, и ram), однако вариант с расширением ram предоставляет дополнительные возможности по определению точки проигрывания файла и даты его воспроизведения.

pnm: //audio.real.com/melody.ra$0:30

В этом примере файл формата RealAudio будет воспроизводиться с тридцатой секунды (запись типа $мин:сек). Дата начала проигрывания определяется конструкцией $dd:hh:mm:ss:t (день : час : минута : секунда : десятая доля секунды).

Также можно вставить в HTML-документ обыкновенную гиперссылку, при нажатии на которую на компьютере пользователя запустится проигрыватель файлов RealAudio: <A href="audio/melody.ra"><t>файл в формате RealAudio</A>

При встраивании файлов RealAudio с помощью тега <EMBED> необходимо помнить, что расширение файла должно быть не ram, а rpm (для активизации подключаемого к браузеру модуля воспроизведения звуковых файлов RealAudio). Пример встраивания файлов RealAudio в HTML-документ:

```
<HTML> <HEAD>
<TITLE>Встраивание файлов RealAudio в HTML-документ</TITLE>
</HEAD>
<BODY>
<EMBED src="audio/melody.rpm" width="300" height="128" autostart="true" controls="all">
```

</BODY>
</HTML>

Параметры тега <EMBED> для воспроизведения звуковых файлов в формате RealAudio:

- src – указание пути к воспроизводимому звуковому файлу формата RealAudio. Является обязательным параметром;
- width – ширина панели воспроизведения (обязательный параметр);
- height – высота панели воспроизведения (обязательный параметр);
- controls – набор элементов управления на панели воспроизведения;
- autostart – при значении true проигрывание файла начинается автоматически (порядок загрузки и последовательность воспроизведения нескольких файлов в формате RealAudio зависят от настроек браузера и программы-сервера);
- console – создание нескольких панелей воспроизведения;
- nolabels – при значении true вывод информации о проигрываемом файле RealAudio (автор, название и пр.) будет запрещен.

Атрибут controls при загрузке файлом в формате RealAudio может иметь следующие значения:

- all – значение по умолчанию – отображаются все элементы управления;
- controlpanel – кнопки воспроизведения и остановки, индикатор воспроизведения файла и регулятор громкости;
- infopanel – вывод информации о файле;
- infoVolumePanel – вывод информации о файле и регулятор громкости;
- statusBar – окно состояния воспроизведения файла, указание времени и продолжительности файла;
- playButton – только кнопка воспроизведения/паузы;
- stopButton – только кнопка остановки;
- volumeSlider – только регулятор громкости;
- positionField – время воспроизведения и продолжительность файла в строке состояния;
- positionslider – только индикатор текущего положения файла;
- statusField – текстовые сообщения в строке состояния.

2.13.4. Фоновый звук

В завершение разговора о музыкальных форматах, используемых в современном Интернете, следует упомянуть о такой возможности языка разметки HTML, как указание фонового звука для электронного документа:

<BODY bgsound="audio/intro.wav" loop="infinite">.

Данный пример задает для страницы фоновый звук "intro.wav" с помощью параметра bgsound. Конструкция loop="infinite" дает браузеру команду проигрывать указанный файл бесконечное количество раз. Для ограниченного количества воспроизведения фонового звука в качестве значения атрибута loop необходимо использовать целые числа (1, 2, 3 и т.д.).

Контрольные вопросы

1. Какие языки называются языками разметки?
2. Что является главной конструкцией языка HTML?
3. Что определяют атрибуты и их значения?
4. Каким общим правилам подчиняется синтаксис тегов?
5. Обязательным ли является использование атрибутов?
6. Что означает открывающий и закрывающий теги?
7. Каким общим правилам подчиняется синтаксис атрибутов?
8. Каким общим правилам подчиняется синтаксис значений?
9. Что такое "специальные символы"?
10. Опишите структуру HTML-страницы.
11. Для чего предназначен заголовок страницы, и какая информация в него входит?
12. Для каких целей можно использовать META-тег следующего вида: <META HTTP-EQUIV = "Content-type" CONTENT = "text/html; CHARSET = windows-1251">?
13. Какие META-теги используются для описания поискового образа документа?
14. Для чего предназначено тело страницы?
15. Какие элементы можно разместить в теле страницы?
16. Запишите теги, используемые при форматировании текста.
17. Чем определяется уровень заголовка текста?
18. Чем отличается переход к новому абзацу и переход к новой строке?
19. Какие основные текстовые стили реализуются в HTML? Какие для этого используются теги и атрибуты?
20. Запишите код, позволяющий выровнять абзац текста по ширине.
21. Перечислите общие теги списков. Чем создание упорядоченного списка отличается от создания маркированного списка?
22. Каким образом можно изменить порядок нумерации упорядоченного списка и внешний вид маркера?
23. Какой тег реализует создание гиперссылки? Какие обязательные атрибуты в него входят?
24. Чем относительная ссылка отличается от абсолютной?
25. В чем особенность создание внутренней ссылки?
26. Каким образом можно разместить на странице изображение?

27. Опишите результат, полученный при выполнении тега:
<IMG src="foto.jpg" width="234" height="465" align="right" border="2">.
28. Запишите тег, позволяющий вставить на страницу кнопку.
29. Какие атрибуты позволяют задавать цвет и фон страницы? В каком теге их необходимо разместить?
30. Для каких целей используются таблицы?
31. Какой тег служит для определения таблицы?
32. Назовите задачи тегов <TR> и <TD>.
33. Какая таблица будет построена в результате выполнения следующего фрагмента кода?

```
<TABLE BORDER=2 WIDTH="75%">>
    <TR>
        <TD>Текст 1</TD>
        <TD ROWSPAN=2>Текст 2</TD>
        <TD>текст 3</TD>
    </TR>
    <TR>
        <TD>Текст 4</TD> <TD>Текст 5</TD>
    </TR>
</TABLE>
```

34. Что такое формы?
35. Какие основные элементы используются в формах? Дайте характеристику каждого элемента.

3. ТЕХНОЛОГИИ КАСКАДНЫХ ТАБЛИЦ СТИЛЕЙ

Дизайн Web-узлов – это точное размещение компонентов HTML-страниц относительно друг друга в рабочей области окна браузера.

Неточность данного определения Web-дизайна очевидна. В нем не учтены ни цвет, ни форма, ни другие свойства компонентов HTML-страниц. Главное в этом определении – показать ограниченность возможностей HTML-разметки. Позиционирование компонентов на странице является одним из самых слабых мест в HTML.

Компонентами страницы являются: блоки текста, графика и встроенные в страницу приложения. Размер и границы каждого из этих компонентов в рамках HTML-разметки задаются с разной степенью точности. Размер графики и приложений можно задать с точностью до пикселя. Размеры текстовых блоков в HTML задать нельзя. Они вычисляются браузером на основе относительного размера шрифта умолчания.

Автор страницы не может заранее определить настроек браузера пользователя, что существенно ограничивает число вариантов представления информации на странице.

Спецификация **CSS**(Cascading Style Sheets) позволяет остаться в рамках декларативного характера разметки страницы и дает полный контроль над формой представления элементов HTML-разметки.

Каскадные таблицы стилей призваны разрешить противоречие между точностью определения размеров картинок и приложений, с одной стороны, и точностью определения размеров блоков текста и его начертания, с другой.

Кроме размера компонентов, таблицы стилей позволяют определить цвет и начертание текстового фрагмента, изменять эти параметры внутри текстового блока, выполнять выравнивание текстового блока относительно других блоков и компонентов страницы.

Наличие всех этих возможностей позволяет говорить о CSS как о средстве разделения логической структуры документа и формы его представления. Логическая структура документа определяется элементами HTML-разметки, в то время как форма представления каждого из этих элементов задается CSS-описателем элемента.

CSS позволяет полностью переопределить форму представления элемента разметки по умолчанию. Например, ... определяет отображение текста курсивом:

```
<i>Отобразим текст курсивом</i>
```

В результате на странице получим:

Отобразим текст курсивом

А теперь переопределим стиль отображения для элемента разметки i:

```
<i style="text-decoration:underline;font-style:normal;">
Отобразим текст курсивом
</i>
```

Результат показан на рис. 22.

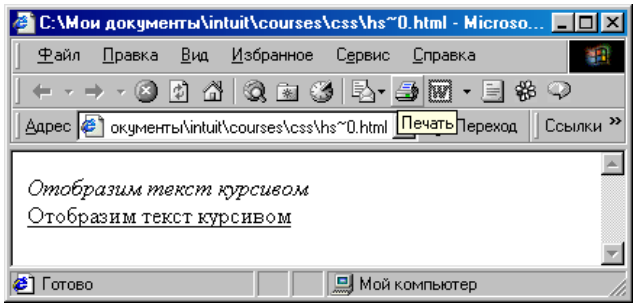

Рис. 22

Этот пример показывает, что привычный стиль отображения элементов может быть полностью изменен при помощи CSS. В данной технологии HTML-разметка носит чисто декларативный характер.

Практическая значимость CSS для Web-инжиниринга заключена в том, что процесс создания узла можно формализовать и представить в виде следующей последовательности действий.

- Сначала нужно определиться с номенклатурой страниц, т.е. все страницы проектируемого Web-узла разбить на типы, например: домашняя страница, навигационные страницы, информационные страницы, коммуникационные страницы и т.п. У каждого узла этот перечень может быть своим.
- Для каждого из типов страниц разрабатывается определенная логическая структура (стандартный набор компонентов страницы).
- После этого разрабатывается навигационная карта узла и форма ее реализации на страницах.
- Для каждого стандартного компонента страницы разрабатывается стиль его отображения (CSS-описатель).
- Теперь остается только рисовать картинки, создавать анимацию, писать программы, вручную вводить текст и графику или генерировать содержание страниц автоматически во время обращения к ним.

3.1. СПОСОБЫ ПРИМЕНЕНИЯ CSS

Под способами применением CSS будем понимать форму декларирования стиля на HTML-странице и форму связывания описания стиля отображения элемента разметки с самим этим элементом. Речь идет о том, где и в какой форме автор страницы (или дизайнер) описывает стиль и как и в какой форме на него ссылается.

Итак, различают три способа применения стилей:

- переопределение стиля в элементе разметки (Inline Style Sheet – внутренние таблицы стилей);
- размещение описания стиля в заголовке документа в элементе STYLE (глобальные таблицы стилей);
- размещение ссылки на внешнее описание через элемент LINK (связанные таблицы стилей).

3.1.1. Переопределение стиля

Под переопределением стиля в элементе разметки мы понимаем применение атрибута STYLE у данного элемента разметки:

```
<h1 style="font-weight:normal; font-style:italic; font-size:10pt;">
Заголовок первого уровня
</h1>
```

Атрибут style можно применить внутри любого элемента разметки. Например, мы можем через style определить ширину и выравнивание элемента hr (горизонтальное отчеркивание):

```
<hr style="width:100px;">
```

Очевидно, что не всякие параметры стиля можно установить для конкретного элемента разметки. Стили разработаны в первую очередь для управления отображением текста. Не следует увлекаться стилями при управлении отображением нетекстовых элементов HTML-разметки.

3.1.2. Элемент STYLE

Применение элемента STYLE – это основной способ внедрения каскадных таблиц стилей в ткань HTML-документа. Кроме управления отображением элементов разметки, элемент STYLE позволяет описывать стилевые свойства элементов, которые можно изменять при программировании на JavaScript.

Элемент STYLE позволяет определить стиль отображения для:

- стандартных элементов HTML-разметки;
- произвольных классов (селектор class);
- HTML-объектов (селектор id).

К сожалению, работа с селекторами в браузерах различных производителей может преподнести различного рода сюрпризы. Особенно это касается работы с селектором ID.

Стандартные элементы разметки описываются в элементе STYLE следующим способом:

```
<head>
<style>
p {color:darkred;text-align:justify;font-size:8pt;} </style>
</head>
<body>
...
<p>
…
</p>
...
</body>
```

Теперь все параграфы документа будут отображаться стилем из элемента STYLE, если только стиль не будет переопределен каким-либо способом. В STYLE можно определить стиль любого элемента разметки.

3.1.3. Ссылка на внешнее описание

Обычно такой способ используют для придания нескольким документам одного стиля. Ссылка на описание стиля, расположенное за пределами документа, осуществляется при помощи элемента LINK, который размещают в элементе HEAD. Внешнее описание может представлять из себя файл, содержание которого – описание стилей. Описание стилей в этом файле будет по синтаксису в точности совпадать с содержанием элемента STYLE.

Ниже приведен пример ссылки на внешнее описание стилей:

<link type="text/css" rel="stylesheet" href="http://kuku.ru/my_css.css">

Важными здесь являются значения атрибутов rel и type. Rel обязан иметь значение "stylesheet". Type может принимать значения: "text/css" или "text/javascript". Второй тип описания стилей введен Netscape.

Атрибут href задает универсальный локатор ресурса (URL) для внешнего файла описания стилей. Это может быть ссылка на файл с любым именем, а не только на файл с расширением *.css.

Файл my_css.css, в свою очередь, является обычным текстовым файлом. Например:

```
body {background:black; font-size:9pt; color:red; font-family:Arial Black;}
.base{color:blue; font-style:italic;}
h1 {color:white;}
#bold {font-weight:bold;}
```

3.2. НАСЛЕДОВАНИЕ И ПЕРЕОПРЕДЕЛЕНИЕ

При обсуждении технических спецификаций часто бывает полезно вникнуть в смысл названия. В названии принято точно определять суть и назначение стандарта или спецификации. Описание стилей отображения элементов HTML-разметки носит название "Каскадные таблицы стилей". Со словом "стилей" все более-менее понятно. Под словом "таблицы" следует понимать набор свойств элемента разметки, который можно представить в виде строки в таблице свойств, т.е. элементы разметки – строки, а свойства – столбцы. А вот слово "каскадные" требует пояснения.

Во-первых, существует иерархия элементов разметки (дерево объектов на странице). Во-вторых, свойства этих объектов могут наследоваться. Таким образом, в дереве объектов образуется ветвь, которая ведет к листу дерева – элементу разметки, например, элементу списка или параграфу. Его свойства определяются элементами разметки, в которые вложен элемент, и описателями стиля для данного элемента (рис. 23).

Текст на рис. 23 закодирован в терминах разделов и списка следующим образом:

<DIV STYLE="margin-left:10px;margin-top:10px;">

Это начало первого раздела, который сдвинут на 10 пикселей вправо относительно левого края параграфа и на 10 пикселей вниз относительно стандартной границы параграфа.

<DIV STYLE="margin-left:10px;margin-top:20px;
text-indent:10px;font-style:italic;">

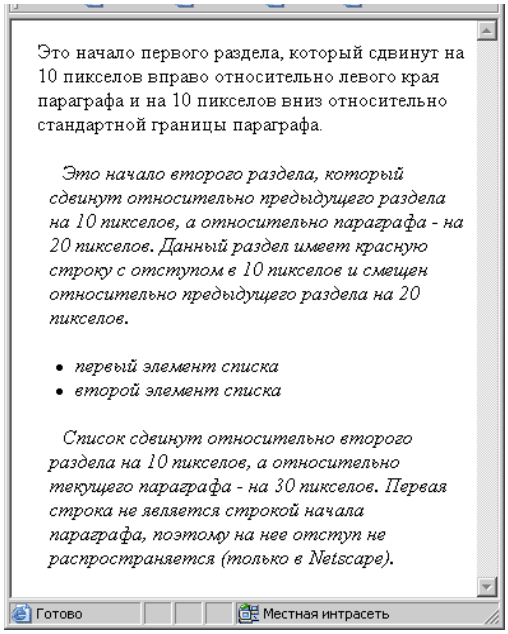

Рис. 23

Это начало второго раздела, который сдвинут относительно предыдущего раздела на 10 пикселей, а относительно параграфа – на 20 пикселей. Данный раздел имеет красную строку с отступом в 10 пикселей и смещен относительно предыдущего раздела на 20 пикселей.

<UL STYLE="margin-left:10px;">
<LI>первый элемент списка
<LI>второй элемент списка
</UL>

Список сдвинут относительно второго раздела на 10 пикселей, а относительно текущего параграфа – на 30 пикселей. Первая строка не является строкой начала параграфа, поэтому на нее отступ не распространяется (только в Netscape).

</DIV>

</DIV>

Таким образом отступы отсчитываются относительно элемента, в который вложен текущий элемент. Все параметры, которые не были переопределены в текущем элементе, наследуются из старшего по иерархии элемента. Последнее хорошо продемонстрировано в применении стилей отображения списка, который вложен в раздел и поэтому отображается курсивом.

При использовании стилей действуют следующие правила старшинства стилей (рис. 24):

- сначала применяются стили браузера по умолчанию;
- стили браузера по умолчанию переопределяются прилинкованными стилями (элемент `LINK` заголовка документа);
- прилинкованные стили переопределяются описаниями стилей в элементе `STYLE`;
- стили элемента `STYLE` переопределяются атрибутом `STYLE` в любом из элементов разметки.

Рис. 24

Не все атрибуты стиля могут наследоваться. Например, "набивка" (отступ содержания элемента от его границ) элемента `BODY` не наследуется вложенными в него элементами и определяется по умолчанию или прописывается для каждого элемента отдельно. Алгоритмы наследования в Internet Explorer и в Netscape Navigator разные, поэтому для единства отображения элементов следует прописывать стиль по максимуму атрибутов.

3.3. СИНТАКСИС ТАБЛИЦ СТИЛЕЙ

Независимо от того, какой способ применения таблиц стилей использован, синтаксис их схож. Он состоит из трех частей:

- Селектора (selector), обладающего свойствами, которые, в свою очередь, имеют значения.
- Свойств (property) селектора. Например, к свойствам селектора абзаца (P) относятся отступы (margin), шрифты (font) и т.д.
- Значений (value) свойств селектора.

Свойства и значения образуют объявление (declaration). Селектор и объявление образуют правило (rule) со следующим порядком записи.

Формально стиль отображения элементов разметки задается ссылкой в элементе разметки на селектор стиля. Синтаксис описания стилей в общем виде представляется следующим образом:

```
selector[, selector[, ...]]
      { attribute:value;
      [attribute:value;...] }
```

или

```
selector selector [selector ...]
       { attribute:value;
       [attribute:value;...] }
```

В первом варианте перечислены селекторы, для которых действует данное описание стиля. Второй вариант задает иерархию вложенности селекторов, для совокупности которых определен стиль. Описания стилей размещаются либо внутри элемента `STYLE`, либо во внешнем файле.

В качестве селектора можно использовать имя элемента разметки, имя класса и идентификатор объекта на HTML-странице.

Атрибут `(attribute)` определяет свойство отображаемого элемента, например левый отступ параграфа `(margin-left)`, а значение `(value)` – значение этого атрибута, например, 10 типографских пунктов (10 pt).

3.3.1. Селектор – имя элемента разметки

Когда автор Web-узла хочет определить общий стиль всех страниц, он просто прописывает стили для всех элементов HTML-разметки, которые будут использоваться на страницах. Это дает возможность скомпоновать страницы из логических элементов, а стиль отображения элементов описать во внешнем файле.

Такой способ создания сайта позволяет автору изменять внешний вид всех страниц путем внесения изменений в файл описания стилей, а не в файлы HTML-страниц.

Внешний файл при этом может выглядеть следующим образом:

```
I, EM {color:#003366;font-style:normal}
A, I {font-style:normal;font-weight:bold;
    text-decoration:line-through}
```

В первой строке этого описания перечислены селекторы-элементы, которые будут отображаться одинаково:

<I>Это курсив</I> и это тоже <EM>курсив</EM>

Последняя строка определяет стиль отображения вложенного в гипертекстовую ссылку курсива:

<A NAME=empty><I>intuit</I></A>

В данном случае переопределение состоит в том, что текст отображается внутри гипертекстовой ссылки перечеркнутым, причем жирным шрифтом.

3.3.2. Селектор – имя класса

Имя класса не является каким-либо стандартным именем элемента HTML-разметки. Оно определяет описание класса элементов разметки, которые будут отображаться одинаково. Для того чтобы отнести элемент разметки к тому или иному классу, нужно воспользоваться его атрибутом `CLASS:`

```
<STYLE>
.test {color:white;background-color:black;}
</STYLE>
...
<P CLASS="test">
Этот параграф мы отобразим белым цветом по черному фону
</P>
...
<P>
Эту <A CLASS="test">гипертекстовую ссылку</A>
мы отобразим белым цветом по черному фону.
</P>
```

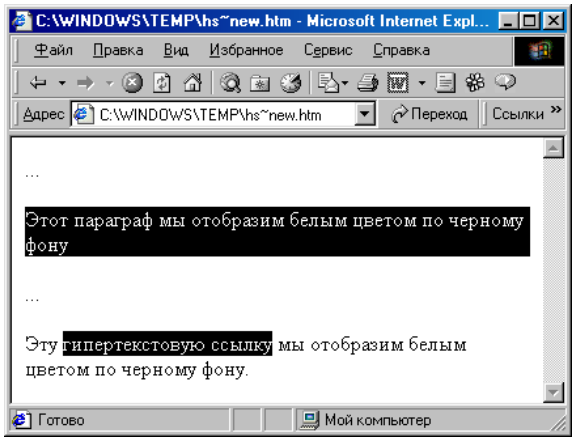

Рис. 25

Таким образом, в любом элементе разметки можно сослаться на описание класса отображения. При этом совершенно необязательно, чтобы элементы разметки были однотипными. В примере к одному классу отнесены и параграф, и гипертекстовая ссылка в другом параграфе.

Лидирующую точку в имени класса можно опустить. Она задается из соображений сохранения единства описания. Например, можно определить классы отображения однотипных элементов разметки:

```
a.menu { color:red;background-color:white;  text-decoration:none; }
a.paragraph { color:navy; text-decoration:underline; }
```

В данном примере класс гипертекстовых ссылок `menu` имеет одно описание стиля, а класс гипертекстовых ссылок `paragraph` – совершенно другое. При этом каждый из этих классов нельзя применить к другим элементам разметки, например, параграфу или списку. Если имя элемента разметки не задано, это означает, что класс можно отнести к любому элементу разметки – корневой класс описания стилей. Это очень похоже на обозначение имени корневого домена в системе доменных имен. Собственно ничего удивительного здесь нет, так как система классов объектов на HTML-странице представляет собой дерево. Элементы разметки – это узлы дерева.

3.3.3. Селектор – идентификатор объекта

Объектная модель документа (Document Object Model) описывает документ как дерево объектов. Объектами являются: сам документ, его разделы (элемент `DIV`), картинки, параграфы, приложения и т.п. Каждый из объектов можно поименовать и обращаться к нему по имени. Данная возможность используется при программировании страниц на стороне клиента.

Применение идентификатора объекта оправдано еще и в случае модификации атрибута описания стиля для данного объекта в его CSS-описании. Вместо двух описаний классов, которые отличаются только одним из параметров, можно создать одно описание класса и описание идентификатора объекта. Описание стиля для объекта задается строкой, в которой селектор представляет собой имя этого объекта с лидирующим символом `"#"`:

```
a.mainlink { color:darkred; text-decoration:underline;  font-style:italic; }
#blue { color:#003366 }
...
```

<A CLASS=mainlink>основная гипертекстовая ссылка</A>
<A CLASS=mainlink ID=blue>модифицированная гипертекстовая ссылка</A>

Следует отметить, что интерпретация идентификаторов объектов в Internet Explorer и Netscape Navigator различна. Существует еще атрибут name у элемента разметки. При идентификации объекта Netscape Navigator обычно имеет дело именно с этим атрибутом, а Internet Explorer – с атрибутом `ID`.

Различия в интерпретации `ID` в браузерах при декларативном использовании CSS не очень страшны. Другое дело, если автор решится программировать стили, т.е. изменять значения атрибутов описателей стилей. В этом случае разница объектных моделей документов в Netscape Navigator и Internet Explorer проявится в полной мере. Фактически, придется для каждого из браузеров разрабатывать совершенно разные страницы.

3.3.4. Псевдоклассы и псевдоэлементы

Синтаксис:
селектор:псевдокласс { свойства }
селектор.класс:псевдокласс { свойства }
селектор:псевдоэлемент { свойства }
селектор.класс:псевдоэлемент { свойства }

Псевдоклассы и псевдоэлементы – это особые классы и элементы, присущие CSS и автоматически определяемые поддерживающими CSS браузерами. Псевдоклассы различают разные типы одного элемента, создавая при определении собственные стили для каждого из них. Псевдоэлементы являются частями других элементов, задавая этим частям отличный от элемента в целом стиль.

Список псевдоклассов и псевдоэлементов

Anchor Pseudo Classes – эти псевдоклассы элемента <a href=" ">, обозначающего ссылку. Псевдоклассы этого элемента:

:link (еще не посещенная ссылка);
:active (активная ссылка);
:visited (посещенный ранее URL);
:hover (псевдокласс, возникающий при поднесении курсора к ссылке).

Другие псевдоклассы:

:first-line. Этот псевдоэлемент может быть использован с block-level элементами (p, h1 и т.д.). Он изменяет стиль первой строки этих элементов.
:first-letter. Изменяет первый символ абзаца.
:focus. Элемент, имеющий фокус ввода.
:lang. Этот псевдокласс определяет текущий язык.
:before. Определяет содержимое перед элементом.
:after. Определяет содержимое после элемента.

П р и м е р :

```
a:link,a:visited {color:blue}
a:active {color:red}
a:hover {text-decoration:none}
```

В данном примере все элементы Anchor (ссылки) будут синими. При нажатии (в активном состоянии) поменяют цвет на красный. И при подведении курсора мышки исчезнет подчеркивание.

3.3.5. Меры длины

Синтаксис:
[число] плюс [единица измерения] (без пропусков)
П р и м е р :
566pt
Единицы длины:
px – pixels, пикселы
in – inches, дюймы
cm – centimeters, сантиметры
mm – millimeters, миллиметры
pt – points, точка (1pt = 1/72in)
pc – picas (1pc = 12pt)
% – процент

3.4. БЛОЧНЫЕ И СТРОКОВЫЕ ЭЛЕМЕНТЫ

В описании элементов разметки языка HTML существует понятие строкового (inline) элемента разметки и блочного (block) элемента разметки. Формально они определены в DTD (Document Type Definition) SGML-описания (Standard Generalised Markup Language) языка HTML. Проще всего объяснить различие между блоком и строковым элементом можно на примере:

параграф – это блочный элемент разметки;
выделение курсивом – это строковый элемент разметки.

Совершенно естественно, что по набору атрибутов управления отображением (атрибуты описания стиля) строковые и блочные элементы отличаются. Упрощенно можно сказать, что атрибуты описания стиля строкового элемента являются подмножеством атрибутов описания стиля блочного элемента.

Обобщениями блочного и строкового элементов с точки зрения стилей выступают элементы DIV и SPAN соответственно.

DIV играет роль универсального блока. Блочный элемент всегда отделен от прочих элементов страницы (контекста) пустой строкой. DIV не несет никакой смысловой нагрузки. Часто говорят, что DIV – это раздел страницы. Но на самом деле его применение имеет смысл только в контексте CSS. Никаких правил по умолчанию для отображения DIV не существует. Это просто новая строка текста.

DIV позволяет применить атрибуты стиля, связанные с границей блока и отступами блока от границ старшего элемента, а также "набивку", т.е. отступ от границы блока до границы вложенного элемента (рис. 26):

<DIV STYLE="margin:20px;padding:10px;">

Блочный элемент, заданный элементом разметки DIV.

<P>Для него определена граница и отступы как от границ старшего элемента разметки, так и для вложенных в него элементов разметки.</P>

</DIV>

В данном примере внутри окна браузера расположен блочный элемент (DIV), внутрь которого помещен еще один блочный элемент (P). DIV имеет белый фон и границу.

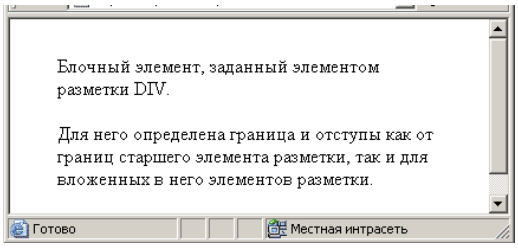

Рис. 26

Если текст будет просматриваться браузерами, не поддерживающими CSS, элемент DIV использовать не рекомендуется. В этом случае лучше применить параграф или другой подходящий по смыслу элемент разметки из стандартного набора HTML.

Элемент разметки SPAN – это обобщенный строковый элемент разметки, применение которого не приводит к образованию блока текста. Он может заменить элементы FONT, I, B, U, SUB, SUP и т.п. Приведем примеры таких соответствий:

HTML-элемент	**CSS-аналог**
 ... 	...
...	...
...	...
<u>...</u>	...

В новых версиях браузера Netscape описания строковых стилей пересекаться не должны. Тег конца элемента строкового типа закрывает ближайший элемент, а не тот, который открыт тегом начала данного строкового стиля. Также и в случае применения элемента SPAN, где тег конца можно соотнести только с ближайшим тегом начала элемента SPAN:

<B>предложение <I>с пересекающимися</B> стилями</I>

Результат выполнения этого кода представлен на рис. 27.

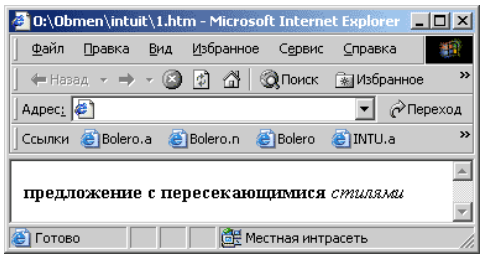

Рис. 27

<SPAN STYLE="font-weight:bold;">предложение <SPAN STYLE= "font-style:italic;"> с пересекающимися </SPAN> стилями</SPAN>

Этот код приводит к результату, показанному на рис. 28.

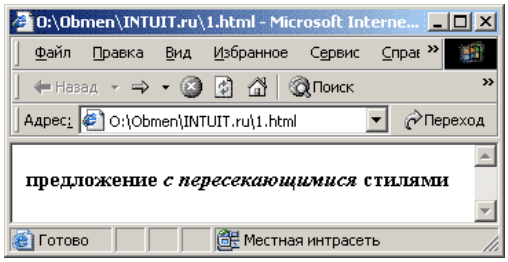

Рис. 28

Применение элемента SPAN ограничено браузерами, которые поддерживают CSS. При этом не все атрибуты спецификации CSS поддерживаются в браузерах. Например, атрибут vertical-align, который призван заменить элементы SUP и SUB, не поддерживается ни одним из браузеров.

3.5. СВОЙСТВА БЛОКОВ

Блочные элементы (блоки текста или box) позволяют оперировать с текстом в терминах прямоугольников, которые этот текст занимает. При этом блок текста становится элементом дизайна страницы с теми же свойствами, что и картинка, таблица или прямоугольная область приложения.

Блок текста обладает свойствами: высоты (height), ширины (width), границы (border), отступа (margin), набивки (padding), произвольного размещения (float), управления обтеканием (clear).

Графически свойства представлены на рис. 29 следующим образом.

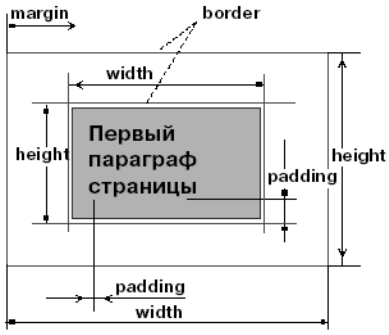

Рис. 29

Задаваться ширина и высота блока могут в типографских пунктах (pt), пикселах (px) и условных единицах (em) (рис. 30):

```
<DIV STYLE="width:200px;">пикселы</DIV>
<DIV STYLE="width:200pt;">типографские пункты
</DIV>
<DIV STYLE="width:5em;">условные единицы</DIV>
```

С высотой блока текста следует быть осторожным, так как в четвертой версии Netscape Navigator многие из атрибутов CSS не поддерживаются, в том числе высота обычного блочного элемента.

Расстояние от границы блочного элемента до границы вложенного в него блочного элемента называется padding. Для обозначения этого свойства будем использовать слово "набивка" или словосочетание "внутренний отступ".

Отступ от "набивки" внешнего блочного элемента до границы вложенного элемента называется margin. Для его обозначения мы будем употреблять термин "отступ" или словосочетание "внешний отступ".

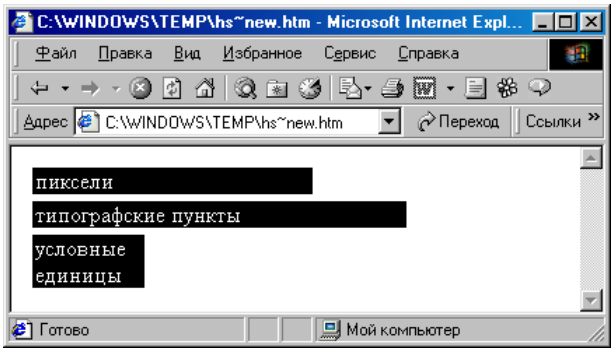

Рис. 30

Таким образом, padding и margin характеризуют отступы блочного элемента относительного начала его содержания и относительно границы охватывающего его элемента разметки, соответственно (рис. 31).

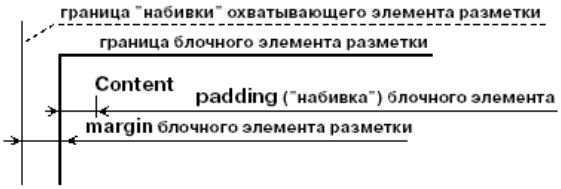

Рис. 31

Отступы и "набивка" могут быть левыми, правыми, верхними и нижними. CSS позволяет изменять любые из них.

При отображении блока текста можно показать его видимую границу. CSS позволяет определить ее стиль, ширину и цвет. При применении видимой границы следует учитывать специфику браузеров. Одним из осмысленных способов применения границы является видимое ограничение "плавающих" блоков текста.

"Плавающий" текстовый блок позволяет реализовать возможность обтекания этого блока текстом.

Обтекание одного текста другим происходит в том же самом ключе, что и обтекание текстом картинки или таблицы. Текст охватывающего блока стремится втиснуться в свободное место, оставленное "плавающим" блоком. Когда граница "плавающего" блока кончается, охватывающий блок распространяется на всю ширину отведенного для текста пространства.

Таким образом, блок текста с точки зрения размещения на странице равноценен картинкам или прямоугольным областям приложений.

3.6. ОТСТУПЫ (MARGIN)

При отображении блока текста на листе бумаги вокруг него обычно оставляют поля. Поля есть и в ученических тетрадях, и в больших серьезных книгах. Поля можно задавать либо относительно границы страницы, либо относительно самого блока текста. В первом случае мы имеем дело с "набивкой" (padding), а во втором – с отступом (margin). Собственно, ширина поля будет определяться суммой ширины "набивки" и ширины отступа (рис. 32).

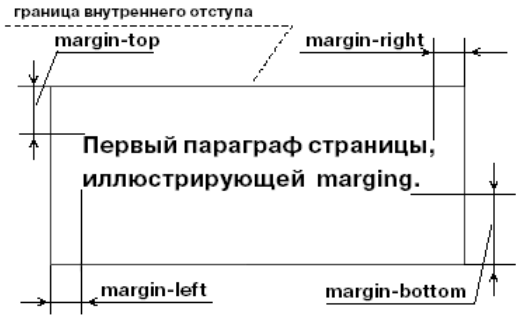

Рис. 32

Обычно пунктирная линия и граница блока являются невидимыми линиями. Они угадываются по выровненному краю текста. Вернее угадывается суммарная ширина полей. Стрелки указывают направление отсчета отступа. Padding отсчитывается от внешней границы блока внутрь блока, в то время как margin – от внешней границы блока в область охватывающего его блока (наружу).

Внешний отступ (margin) может отсчитываться по любому направлению относительно сторон блока:

margin-left – левый внешний отступ. Определяет расстояние от левой границы блока текста до левой границы внутреннего отступа ("набивки", padding) охватывающего элемента;

margin-right – правый внешний отступ. Определяет расстояние от правой границы блока текста до правой границы внутреннего отступа ("набивки", padding) охватывающего элемента;

margin-top – верхний внешний отступ. Определяет расстояние от верхней границы блока текста до верхней границы внутреннего отступа ("набивки", padding) охватывающего элемента;

margin-bottom – нижний внешний отступ. Определяет расстояние от нижней границы блока текста до нижней границы внутреннего отступа ("набивки", padding) охватывающего элемента;

margin – задает общий внешний отступ от всех сторон блока текста. Применяется в том случае, если блок текста равноудален от всех границ внутреннего отступа охватывающего элемента. Также в этом элементе возможна группировка значений. При этом значения указываются в следующем порядке:

– если указано одно значение, то оно присваивается сразу всем полям;

– если указаны два значения, то первое присваивается верхнему и нижнему полю, а второе – левому и правому;

– если указаны три значения, то первое значение присваивается верхнему полю, второе – левому и правому, третье – нижнему.

Графически эти отступы можно представить следующим образом (рис. 33).

Рис. 33

В данном случае для параграфа использовалось следующее описание стиля:

P{margin-left:50px;margin-right:5px; margin-top:15px;margin-bottom: 50px; padding:0px;text-align:left; }

Нужно иметь в виду, что браузеры могут отображать эти параметры по-разному.

Если размер всех внешних отступов одинаковый, то можно просто воспользоваться атрибутом `margin`:

P { margin:5px; }

При применении внешнего отступа следует помнить, что он отсчитывается от границы элемента до границы внутреннего отступа ("набивки", padding) охватывающего элемента. Если этот факт не учитывать, то общая ширина видимых полей может оказаться больше, чем указано во внешнем отступе.

3.7. НАБИВКА (PADDING)

Текст внутри блока начинается не от самой его границы. Между границей и содержанием блока есть свободное пространство. Оно называется внутренний отступ текстового блока или padding. Совместно с внешним отступом (margin) текстового блока padding образует общее поле отступа от границы охватывающего блок элемента разметки.

Padding можно проиллюстрировать на примере левого внутреннего отступа текста в параграфе (рис. 34).

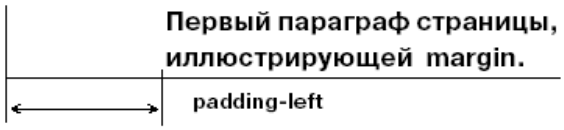

Рис. 34

Для этого примера при описании параграфа использовался стиль:

P { padding-left:100px;text-align:left; border-width:1px; }

У блока текста существует четыре стороны. Соответственно различают:

padding-left – левый внутренний отступ, который определяет расстояние от левого края блока до его содержания;

padding-right – правый внутренний отступ, который определяет расстояние от правого края блока до его содержания;

padding-top – верхний внутренний отступ, который определяет расстояние от верхнего края блока до его содержания;

padding-bottom – нижний внутренний отступ, который определяет расстояние от нижнего края блока до его содержания;

padding – определяет единый размер внутреннего отступа блока. Этот параметр задается в случае одинакового размера отступа от всех сторон блока. Также в этом элементе возможна группировка значений. При этом значения указываются в следующем порядке:

– если указано одно значение, то оно присваивается сразу всем отступам;

– если указаны два значения, то первое присваивается верхнему и нижнему отступу, а второе – левому и правому;

– если указаны три значения, то первое значение присваивается верхнему отступу, второе – левому и правому, третье – нижнему.

Проиллюстрируем применение padding на примере (рис. 35):

P { padding-left:100px;padding-right:50px; padding-top:20px; padding-bottom:10px; text-align:left;border-width:1px; }

Рис. 35

При установке padding следует помнить, что этот параметр задает размер отступа от границы блока до границы внешнего отступа (margin) содержания блока. По этой причине общий размер поля может оказаться больше, чем задано в параметре padding.

3.8. ГРАНИЦА (BORDER)

У каждого блочного элемента разметки есть граница. От границы отсчитываются отступы (margin и padding). Вдоль границы "плавающего" блока его обтекает текст.

Для описания границ блоков применяются следующие атрибуты:

border-top-width – ширина верхней границы блока;

border-bottom-width – ширина нижней границы блока;

border-left-width – ширина левой границы блока;

border-right-width – ширина правой границы блока;

border-width – ширина границы блока. Задается в том случае, если ширина границы блока одинаковая по всему периметру блока. При этом значения указываются в следующем порядке:

– если указано одно значение, то оно присваивается сразу всем сторонам рамки;

– если указаны два значения, то первое присваивается верхней и нижней стороне, а второе – левой и правой;

– если указаны три значения, то первое значение присваивается верхней стороне, второе – левой и правой, третье – нижней;

border-color – цвет границы блока. Согласно спецификации CSS1, может быть задан для каждой из границ блока. Например, `border-right-color:red`. Может задаваться как мнемоникой (red, blue, navy и т.п.), так и в нотации RGB (`border-color:#003366`). Указание цвета для каждой из границ поддерживается не всеми браузерами;

border-style – тип линии границы блока. Может принимать значения: `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`. Согласно спецификации CSS1, может быть задан для каждой из границ блока. Например, `border-right-style:dotted`. Указание типа линии границы поддерживается не всеми браузерами.

Для описания границы нет необходимости указывать в стиле все атрибуты. Существует сокращенная запись атрибутов. Например, для описания верхней линии границы можно использовать запись типа:

P { border-top:1px dotted red; }

атрибут: ширина_линии тип_линии цвет_линии код

Если необходимо ограничить блок текста границей, то это может выглядеть примерно так (рис. 36).

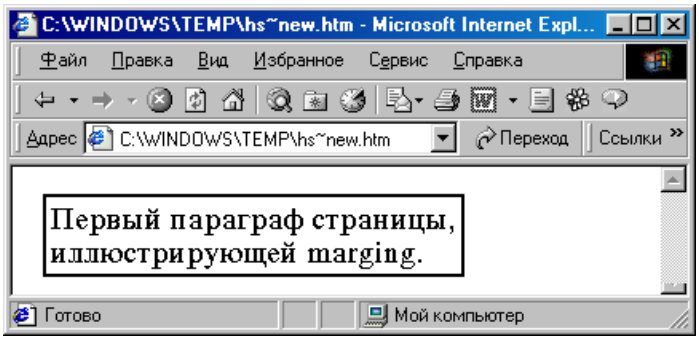

Рис. 36

В этом примере было использовано следующее описание стиля отображения границы:

P {text-align:left;border-width:2px; border-color:darkred;border-style:solid;}

Применение границы для обозначения блока не самый лучший способ оформления документа. Во всяком случае, его применяют нечасто.

Указывая границу в Internet Explorer, нужно обязательно определять ее тип, в противном случае она не будет отображаться.

3.9. ОБТЕКАНИЕ БЛОКА ТЕКСТА

Под обтеканием блока текстом понимают тот же самый эффект, который можно реализовать для графики, когда картинка не разрывает блок текста, а встраивается в него. Текст в этом случае "обтекает" картинку с одной из сторон – там, где есть свободное поле между границей страницы (элемента) и картинкой. "Обтекание" картинки текстом от обычного встраивания картинки в текст документа отличается тем, что вдоль вертикальной границы картинки располагается несколько строк текста, а не одна.

"Обтеканием" блока текста другим текстом управляют два атрибута CSS: float и clear.

Атрибут float определяет плавающий блок текста. Он может принимать значения:

left – блок прижат к левой границе охватывающего блок элемента;

rigth – блок прижат к правой границе охватывающего блок элемента;

both – текст может обтекать блок с обеих сторон.

Проиллюстрировать обтекание можно на следующем примере (рис. 37).

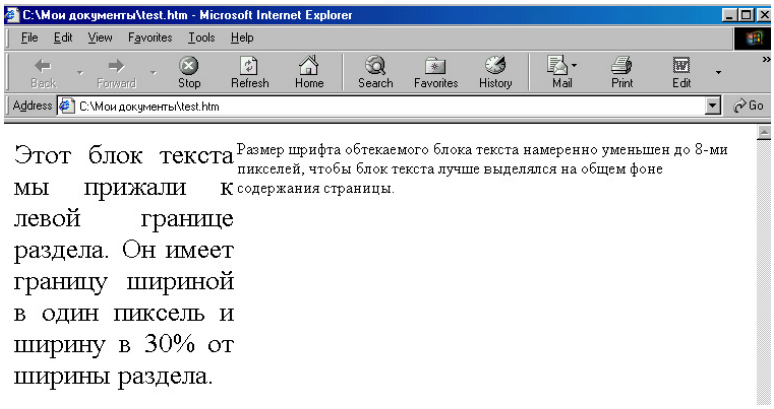

Рис. 37

Для данного примера использовался следующий код:

Этот блок текста мы прижали к левой границе раздела. Он имеет границу шириной в один пиксель и ширину в 30 % от ширины раздела.

 Размер шрифта обтекаемого блока текста намеренно уменьшен до 8-ми пикселей, чтобы блок текста лучше выделялся на общем фоне содержания страницы.

При использовании значения "right" блок текста будет прижат вправо.

Второй атрибут описания стилей clear позволяет управлять собственно обтеканием. Он запрещает наличие плавающих блоков около блока текста. Атрибут может принимать значения: right, left, none, both.

3.10. УПРАВЛЕНИЕ ЦВЕТОМ В CSS

Каскадные таблицы стилей (CSS) в первую очередь описывают свойства текста. Это касается как текстовых блоков, так и строковых элементов разметки содержания страницы. В данном разделе речь пойдет об управлении отображением цвета текста (color) и цвета фона (`background-color`), на котором отображается текст.

Кроме цвета текста и цвета фона, CSS позволяет определять цвет границы текстового блока (`border-color`).

Атрибуты стилей, которые мы собираемся рассмотреть, согласно спецификации Microsoft, относятся к группе атрибутов Color and Background Properties. Всего в эту группу входит семь атрибутов, шесть из которых определяют свойства фона. Кроме цвета фона и его прозрачности, можно управлять фоновой картинкой (координатами ее размещения и способами повторения). К сожалению, Netscape Navigator большинство из этих атрибутов не поддерживает, поэтому мы не будем рассматривать их подробно.

Интерпретация атрибутов цвета в Netscape Navigator и Internet Explorer различна. В Netscape Navigator фоновый цвет отображается только там, где есть текст, а в Internet Explorer фоновый цвет заливает весь блок или строковый элемент вне зависимости от наличия в нем текста.

3.10.1. Цвет текста

В HTML для управления цветом отображаемого текста используется элемент `FONT`. Его аналогом в CSS является атрибут color. Этот атрибут можно применять как для блочных, так и для строковых элементов разметки.

Рассмотрим в качестве блочного элемента разметки ячейку таблицы (рис. 38):

TD { color:darkred; }

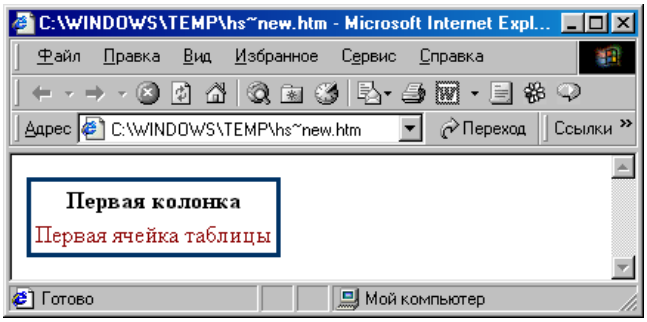

Рис. 38

В данном примере цвет текста определен только для обычной ячейки, поэтому содержание заголовка колонки отображается основным цветом (#003366).

При определении цвета текста для блочного элемента весь текст этого элемента отображается заданным цветом. Частичное изменение цвета возможно, если поместить строковый элемент разметки внутрь блочного (рис. 39):

P { color:darkred; }

I { color:#003366;font-style:normal; }

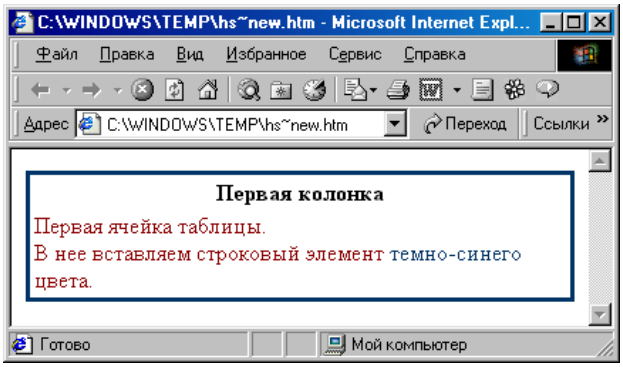

Рис. 39

В данном примере в качестве блочного элемента мы используем параграф, а в качестве строкового элемента (in-line) применяем `I`. Таблица в данном случае большого значения не имеет, но используется для единообразия с предыдущим примером. В нее мы помещаем параграф со встроенным в него in-line элементом разметки.

3.10.2. Цвет фона текста

В HTML цветом фона можно управлять только для конкретного блочного элемента разметки. Таким элементом может быть вся страница:

<BODY BGCOLOR=...>...</BODY>

Или, например, таблица:

<TABLE BGCOLOR=...>...</TABLE>

В приведенном ниже примере для выделения текста применено инвертирование цвета фона и цвета текста (рис. 40):

<SPAN STYLE="background-color:black;color:white;">

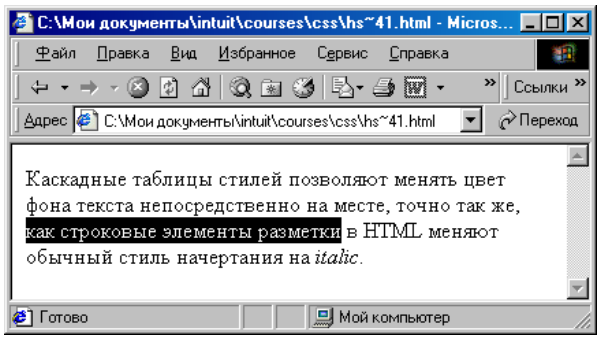

Рис. 40

как строковые элементы разметки

</SPAN>.

При использовании цвета фона следует помнить, что поддержка этого атрибута реализована для всех блочных элементов разметки только в Internet Explorer 4.0. Поддержка CSS в версиях Netscape Navigator гораздо скромнее.

Для работы с фоном элементов существует несколько атрибутов, которые поддерживаются только в Internet Explorer, начиная с версии 4.0: `background-image`; `background-repeat`; `background-attachment`; `background-position`.

- background-image – определяет фоновое изображение элемента.

Возможные значения:

none – фоновое изображение не устанавливается;

URL (+) "имя файла"

П р и м е р :

H1 {background-image:URL("cool.gif");}

- background-repeat – определяет направление, по которому экран заполняется копиями фонового изображения.

Возможные значения:

repeat – повторяет фоновое изображение во всех направлениях (по умолчанию);

repeat-x – повторяет фоновое изображение только по горизонтали;

repeat-y – повторяет фоновое изображение только по вертикали;

no-repeat – не повторяющееся изображение.

П р и м е р :

background-repeat:no-repeat

- background-attachment – определяет, будет ли фоновое изображение зафиксировано в окне браузера или будет прокручиваться вместе с документом.

Возможные значения:

scroll – фоновое изображение прокручивается вместе с содержанием документа;

fixed – не прокручивается.

П р и м е р :

IMG {background-attachment: scroll;}

- background-position – определяет положение фонового элемента относительно левого верхнего угла содержащего его элемента. Одиночные значения устанавливают расстояние по горизонтали и вертикальное смещение по умолчанию, равное 50 %. Возможно сочетание нескольких ключевых слов.

Возможные значения:

отношение (в процентах) смещения фонового изображения от левого края по горизонтали к длине содержащего его элемента (значение по умолчанию 0 %) + отношение (в процентах) смещения фонового изображения от верхнего края по вертикали к высоте содержащего его элемента;

по горизонтали от левого края элемента до фонового изображения + расстояние по вертикали от верхнего края элемента до фонового изображения;

top – размещение фонового изображения по верхнему краю;

middle– размещение фонового изображения по центру элемента;

bottom – размещение фонового изображения по нижнему краю;

left – размещение фонового изображения по левому краю;

center– размещение фонового изображения по центру элемента;

right – размещение фонового изображения по правому краю.

П р и м е р :

background-position:50% 0%

Все свойства фона можно описать в атрибуте `background`:

background:transparent|color url repeat scroll position

П р и м е р :

P { background: gray http://intuit.ru/intuit.gif no-repeat fixed center center; }

Однако при всем изобилии возможностей злоупотреблять ими не стоит.

3.11. ШРИФТ

Шрифтам в компьютерной графике всегда уделялось много внимания. World Wide Web в этом аспекте не является исключением. Но все богатство и разнообразие шрифтов, существующих в природе, для русского языка ограничено фактически тремя шрифтами: serif (обычно Times или другой шрифт с засечками), sans-serif (Arial, или Helvetica, или другой шрифт без засечек) и monospace (Courier). Если быть точным, то здесь перечислены семейства шрифтов. Обычно каждое из этих семейств представлено только одним кириллическим шрифтом.

Автор документа для управления отображением букв текста может применить несколько атрибутов, влияющих на шрифт:

font-family – семейство начертаний шрифта (гарнитура);

font-style – прямое начертание или курсив;

font-weight – "усиление" (насыщенность) шрифта, "жирность" букв;

font-size – размер шрифта (кегль). Задается в пикселях (px) и типографских пунктах (pt);

font-variat – вариант начертания (обычный или мелкими буквами – капитель).

Существует также возможность совместить все эти параметры в одном атрибуте font:

font:bold 12pt sans;

При использовании различных гарнитур (font-famaly) следует помнить, что наличие или отсутствие необходимой автору гарнитуры всецело зависит от предпочтений пользователя. Для кириллицы это может вылиться в появление абракадабры там, где автор применяет отсутствующие у пользователя шрифты.

Самое неприятное, что может возникнуть при использовании шрифтов – это несоответствие моноширинных шрифтов, которые используются в HTML-формах. Обратная связь с пользователем в этом случае оказывается парализованной.

Спецификация CSS разрешает перечислять шрифты в описаниях стилей, что позволяет частично решить проблему подбора шрифта. К сожалению, в Unix и Windows шрифты не согласованы. Фактически, при разработке страниц в CSS используются только классы шрифтов (serif, sans-serif и monospace).

3.11.1. Гарнитура (font-family)

Гарнитура шрифта – это набор начертаний одного шрифта. Шрифт может иметь "прямое" начертание (normal), курсив (italic), "скошенное" (oblique), усиленное по насыщенности ("жирное", bold), "мелкое" (капитель, small-caps) и т.п.

Наиболее распространенные гарнитуры в российской части Web – это Times, Arial, Courier. Причем все они принадлежат к разным группам шрифтов. Times – это пропорциональный шрифт "с засечками" (serif), Arial – это пропорциональный шрифт "без засечек" (sans-serif), а Courier – это моноширинный шрифт (monospace). В Unix вместо Arial чаще применяется Helvetica.

Разницу между этими группами шрифтов лучше всего видно на примере (рис. 41):

 Эта строка набрана пропорциональным шрифтом с засечками.

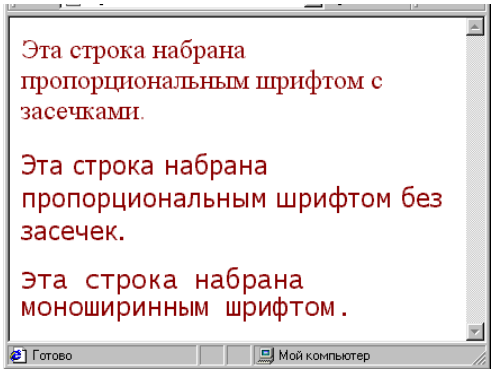

Рис. 41

 Эта строка набрана пропорциональным шрифтом без засечек.

 Эта строка набрана моноширинным шрифтом.

При указании имени группы шрифтов, как это сделано в примере, браузер подбирает наиболее приемлемый для отображения шрифт данной группы из имеющегося набора шрифтов.

Если оптимизация браузера не устраивает автора страницы, то можно указать непосредственно имя гарнитуры шрифта (рис. 42).

```
 <SPAN STYLE="font-family:symbol;  padding-left:65px;">
N</SPAN><BR>
<SPAN STYLE="font-family:symbol; font-size:24px;">
A<SUB>x,y</SUB>=e(y<SUB>a</SUB>+x<SUB>a</SUB>)
</SPAN><BR>
<SPAN STYLE="font-family:symbol; padding-left:60px; font-size:20px;">a=1
</SPAN>
```

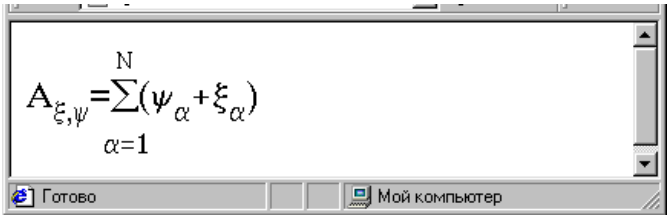

Рис. 42

Лучше всего пользоваться категориями групп шрифтов, тогда текст с точки зрения языка будет везде отображаться правильно. Правда, достигаться эта универсальность будет за счет ограничения возможностей начертания шрифта.

3.11.2. Кегль (font-size)

Кегль – это, если говорить упрощенно, размер шрифта. CSS через параметр font-size позволяет управлять размером букв.

Размер шрифта можно задавать в типографских пунктах (pt, 0,35 мм) или пикселях (px). При установке кегля стоит помнить, что font-size задает не высоту буквы, а размер "очка" под букву, который больше самой буквы.

Вот несколько примеров использования `font-size (рис. 43)`:

```
<P STYLE="font-size:12pt;">
Кегль параграфа установлен в 12 пунктов</P>
<P STYLE="font-size:12px;">
Кегль параграфа установлен в 12 пикселов</P>
<P STYLE="font-size:120%;">
Кегль параграфа установлен в 120 % от размера букв охватывающего параграф элемента</P>
```

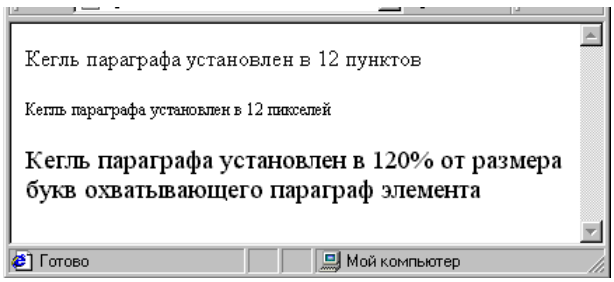

Рис. 43

Кегль можно задавать не только в абсолютных единицах, но и в относительных. Кроме процентов существует еще несколько условных единиц измерения кегля, которые можно применять в CSS (рис. 44):

<P STYLE="font-size:large;"> Размер кегля large</P>
<P STYLE="font-size:small;"> Размер кегля small</P>
<P STYLE="font-size:x-small;"> Размер кегля x-small</P>
<P STYLE="font-size:xx-small;"> Размер кегля xx-small</P>

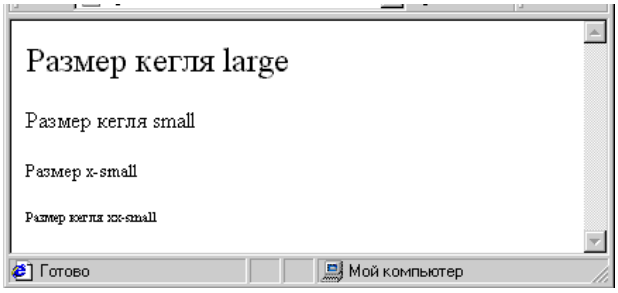

Рис. 44

Аналогично `x-small` и `xx-small`, существуют размеры `x-large` и `xx-large`. Кроме того, есть `larger`, `smaller` и `medium`.

3.11.3. Начертание

У каждой гарнитуры (`font-family`) имеется несколько начертаний. Каждое из них определяется в CSS тремя параметрами стиля: `font-style`, font-variant, `font-weight`.

Атрибут стиля `font-style` определяет прямое начертание (`normal`) и курсив (рис. 45):

<P STYLE="color:darkred;font-style:normal;"> Прямое начертание</P>
<P STYLE="color:darkred;font-style:italic;"> Курсив</P>

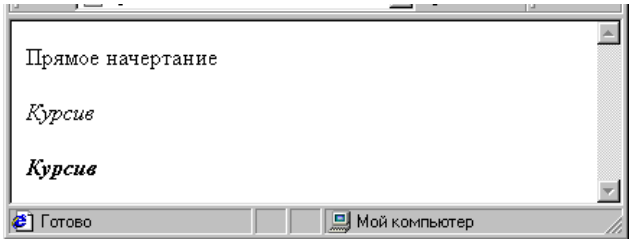

Рис. 45

Если хочется усилить насыщенность ("жирность") шрифта, то в описании стиля указывают атрибут `font-weight`, который принимает значения `normal` или `bold`:

<P STYLE="color:darkred;font-style:italic; font-weight:bold;"> Курсив </P>

Хотя шрифт и масштабируется при помощи изменения кегля, качество его начертания при этом обычно страдает. Для качественного отображения мелких букв в некоторых гарнитурах присутствует начертание капитель. В CSS для использования капители зарезервирован атрибут font-variant, который принимает значения `normal` и `small-caps`. На практике применение font-variant проблематично из-за отсутствия капители в стандартном наборе кириллических шрифтов.

3.12. ТЕКСТ

При обсуждении свойств блочных элементов разметки речь шла о параметрах, относящихся к блоку как целому. Мы не рассматривали внутренние характеристики текста.

Рассказывая о шрифтах, мы акцентировали внимание на начертаниях символов как таковых, а не на их соотношении.

Тем не менее, в стороне остались такие важные характеристики текстового фрагмента, как:

- межбуквенные расстояния;
- высота строк;
- выравнивание;
- отступ в первой строке параграфа;
- преобразования начертания.

Все эти атрибуты сгруппированы в свойства текстовых фрагментов (Text Properties).

3.12.1. Межбуквенные расстояния

Расстояние между буквами автоматически регулируется размером шрифта – кеглем. Чем больше размер шрифта, тем больше расстояние между буквами (рис. 46).

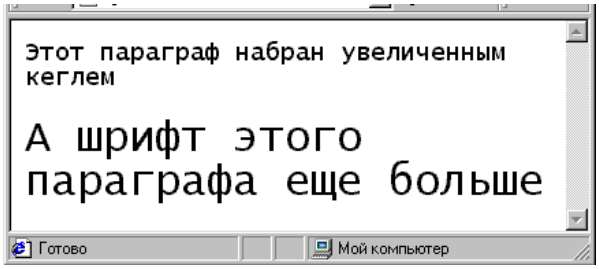

Рис. 46

Нетрудно убедиться, что расстояние между буквами в слове "параграф" первого примера и буквами слова "параграфа" второго примера разное. Во втором случае оно больше.

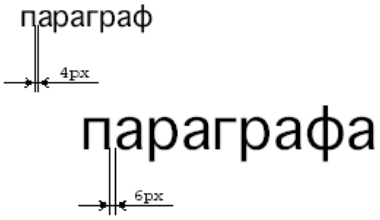

Рис. 47

Моноширинный шрифт выбран не случайно. На пропорциональном шрифте межбуквенное расстояние зависит от начертания букв, и показать его как расстояние между буквами достаточно сложно. У моноширинного шрифта размер символа фиксирован, поэтому и расстояние между буквами прослеживается четко.

Однако не всегда удобно управлять межбуквенным расстоянием через кегль (`font-size`). Бывают случаи, когда нужно либо уплотнить строку, либо увеличить расстояния между буквами. Это можно сделать с помощью атрибута `letter-spacing (рис. 48)`:

```
<P STYLE="font-family:monospace; letter-spacing:5pt; color:black">
Межбуквенное расстояние 5pt</P>
<P STYLE= "font-family:monospace; letter-spacing:10pt; color:black">
Межбуквенное расстояние 10pt </P>
```

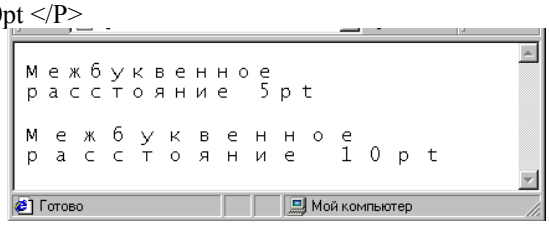

Рис. 48

Правда, в версиях Netscape Navigator 4.x этот параметр не поддерживается.

3.12.2. Выравнивание

По умолчанию все слова в параграфе прижаты влево. Левый край параграфа, таким образом, оказывается выровненным. Точно так же может быть выровнен правый край параграфа или блока текста, и даже оба края вместе.

В обычной HTML-разметке такой эффект достигается за счет применения атрибута *`align`*:

```
<P align=justify>...</P>
```

Аналогичный результат в CSS достигается за счет атрибута `text-align (рис. 49)`:

```
<P STYLE="text-align:right;color:black;">
Этот параграф выровнен по правому краю. Все строки справа кончаются на границе раздела. А вот слева они начинаются с различным  отступом от левого края.</P>
<P STYLE="text-align:justify;color:black;">
```

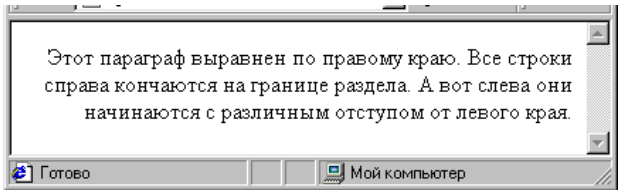

Рис. 49

Этот параграф выровнен по левому и правому краям. Все строки справа кончаются на вертикальной границе раздела. Все строки слева теперь начинаются также с вертикальной границы раздела.</P>

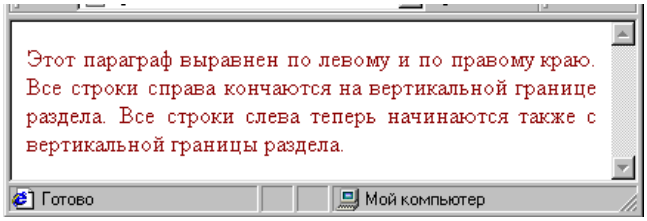

Рис. 50

Выравнивать текст можно в любом блочном элементе. Причем можно не только выравнивать текст по краям блочного элемента, но и центрировать его (`<P STYLE="text-align:center;">...</P>`).

3.12.3. Преобразование шрифта

Преобразование шрифта подразумевает капитализацию слов, перевод всех "больших" и "маленьких" букв в большие, или, наоборот, получение одних строчных.

Рассмотрим несколько примеров (рис. 51):

<P STYLE="text-transform:uppercase;"> Сделать заглавными</P>

<P STYLE="text-transform:lowercase;"> Сделать строчными</P>

<P STYLE="text-transform:capitalize;"> Сделать заглавными первые буквы в словах</P>

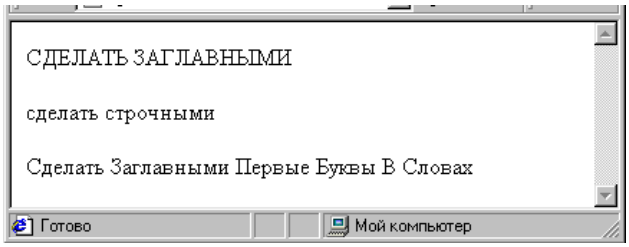

Рис. 51

Выполнение преобразований зависит от алгоритма преобразования символов. В нелокализованных программах переход от строчных букв к прописным осуществляется путем простого смещения по таблице ASCII, что для русского алфавита не приемлемо.

Еще один вид преобразования шрифта – это подчеркивание, перечеркивание или надчеркивание слов. Выполняется такое преобразование с помощью атрибута `text-decoration (рис. 52)`:

<P STYLE="text-decoration:line-through;"> Перечеркнем это предложение.</P>

<P STYLE="text-decoration:underline;"> Подчеркнем это предложение.</P>

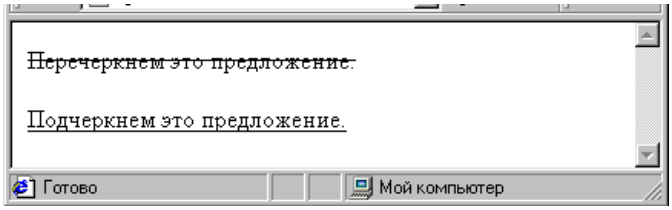

Рис. 52

Для того чтобы преобразование работало, необходимо соответствующее начертание (подчеркнутые или перечеркнутые начертания букв). Очень сложно найти гарнитуру, в которой было бы начертание с надчеркнутыми буквами. Отмена декора происходит, если использовать в text-decoration значение none.

3.12.4. Первая строка параграфа

При оформлении параграфов в технологии CSS автор может воспользоваться "красной" строкой, такую возможность предоставляет ему атрибут `text-indent`.

Речь идет о горизонтальном отступе в первой строке параграфа относительно его левого края (рис. 53):

<P STYLE="text-indent:20pt;"> Этот параграф мы начнем со строки с горизонтальным отступом в двадцать типографских пунктов от левого края параграфа. </P>

<P STYLE="text-indent:-10pt;"> А в этом параграфе мы применим отрицательный горизонтальный отступ в первой строке параграфа.</P>

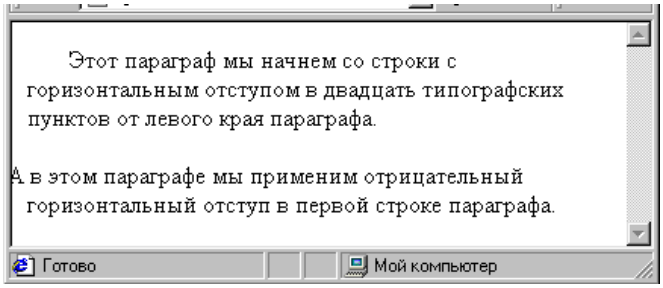

Рис. 53

Отрицательные значения атрибутов – это нормальная практика CSS. Там, где применение отрицательного значения оправдано, например, в случае смещения вложенного блока текста относительно охватывающего элемента разметки, можно указывать отрицательные атрибуты смещения.

Кроме `text-indent` в CSS для оформления первой строки параграфа зарезервирован модификатор стиля `first-line`. Он позволяет не только задать горизонтальное смещение, но и определить другие параметры параграфа:

P:first-line { color:red; }

Еще один параметр, который влияет на отображение первой строки параграфа, – первая буква первой строки. Ее отображением управляет модификатор `first-letter`:

P:first-letter { font-size:20pt; }

К сожалению, оба названных модификатора реализованы не во всех версиях браузеров, поэтому для верности применяют элементы разметки `FONT` и `TABLE`.

3.12.5. Межстрочное расстояние

В CSS межстрочное расстояние определяется параметром `line-height`. Он задает расстояние не между строками, а между базовыми линиями строк. Если, например, взять букву "н" и напечатать ее последовательно друг под другом, то `line-height` будет равно расстоянию между двумя одинаковыми точками букв (рис. 54).

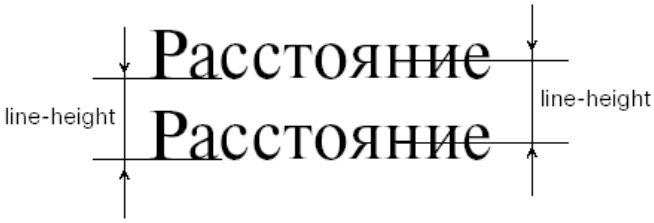

Рис. 54

Посмотрим, как этот параметр влияет на взаимное расположение строк (рис. 55):

```
<P STYLE="line-height:12pt;font-size:12pt; color:black;">
Этот параграф мы набрали кеглем 12 pt. Line-height задан в 12 pt.</p>
<P STYLE="line-height:24pt;font-size:12pt; color:black;">
Этот параграф мы набрали кеглем 12 pt. Line-height задан в 24 pt.</P>
<P STYLE="line-height:6pt;font-size:12pt; color:black;">
Этот параграф мы набрали кеглем 12 pt. Line-height задан в 6 pt.</P>
```

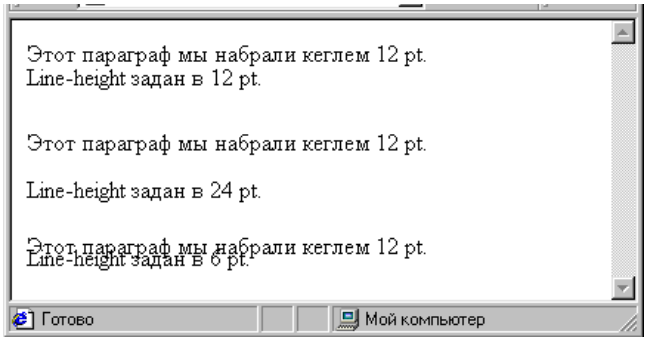

Рис. 55

Первый пример набран со значением `line-height`, равным размеру кегля. Во втором примере значение `line-height` вдвое превышает размер кегля. В третьем примере значение `line-height` в два раза меньше размера кегля – строки стали "наползать" друг на друга.

В связи с использованием line-height следует обратить внимание на применение in-line картинок на HTML-страницах. Под in-line картинкой здесь имеется в виду картинка, которая встроена в тело документа при помощи элемента IMG, но не с новой строки и не как элемент таблицы (рис. 56):

<P STYLE="color:white;background-color:black; font-size:20px;">
В эту строку мы встраиваем картинку – <IMG SRC="inline.gif" BORDER="0" WIDTH="24" HEIGHT="24" ALIGN="top">, на которой изображены концентрические круги.
</P>

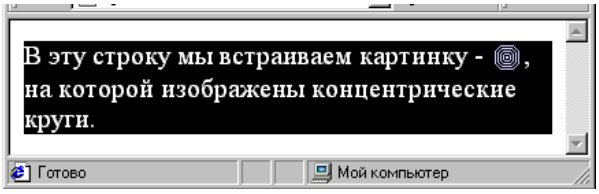

Рис. 56

Картинка имеет размеры 24 × 24 пикселя и выровнена по верхнему краю строки. Ее размер больше размера кегля (20 px), поэтому межстрочное расстояние увеличено браузером автоматически.

<P STYLE="color:white; background-color:black;font-size:24px;">
В эту строку, которая имеет размер кегля 24px, мы встраиваем картинку – <IMG SRC="inline.gif" BORDER="0" WIDTH="24" HEIGHT="24" ALIGN="top"> , на которой изображены концентрические круги.
</P>

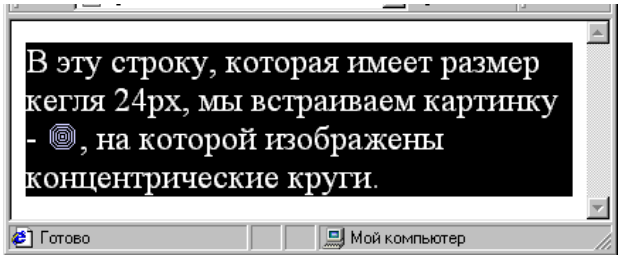

Рис. 57

Таким образом, можно точно позиционировать текст и графику в строке.

3.12.6. Списки

При отображении списков CSS позволяет управлять формой и изображением символа, стоящего перед элементом списка ("пулек" – bullet). Например, в неупорядоченном списке (unordered list) перед элементом списка ставится "жирная" точка:

- первый элемент списка;
- второй элемент списка;
- третий элемент списка.

CSS позволяют управлять формой символов и заменять символы картинками.

Управление отображением элементов списка отнесено к набору свойств, в который входит атрибут display. У этого атрибута может быть только одно значение – none. Если элемент в своем описании имеет атрибут display и этот атрибут равен none, то он не отображается браузером вообще:

<UL STYLE="display:none;">
<LI>Первый элемент списка
<LI>Второй элемент списка
<LI>Третий элемент списка
</UL>

Если посмотреть HTML-код данного документа, то за примером описания списка следует код, который браузер не отобразил.

Атрибут display управляет отображением документа на дисплее компьютера, но не распространяется на другие среды отображения документа.

Форма символа списка в виде "жирной" точки несколько непривычна. Обычно в машинописных документах используют черту. С другой стороны, в рекламных материалах часто в качестве символа списка применяют квадрат или другой символ типографского набора, а также графическую картинку.

CSS позволяет управлять формой символа списка через атрибут list-style-type (рис. 58):

<UL STYLE="list-style-type:square;">
<LI>В виде "пульки" используем квадрат

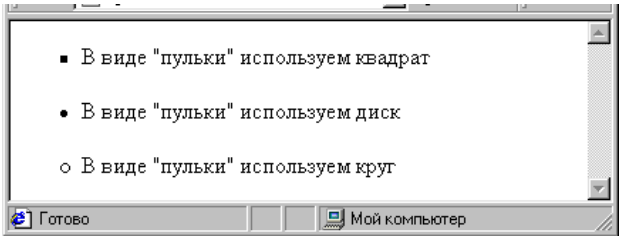

Рис. 58

```
</UL>
<UL STYLE="list-style-type:disk;">
<LI>В виде "пульки" используем диск
</UL> <UL STYLE="list-style-type:circle;">
<LI>В виде "пульки" используем круг
</UL>
```

Управлять отображением "пулек" можно и в упорядоченных списках (`OL`) (рис. 59):

```
<OL STYLE="list-style-type:lower-roman; color:black;">
<LI>...
...
</OL>
<OL STYLE="list-style-type:upper-alpha; color:black;">
<LI>...
...
</OL>
<OL STYLE="list-style-type:lower-alpha; color:black;">
<LI>...
...
</OL>
```

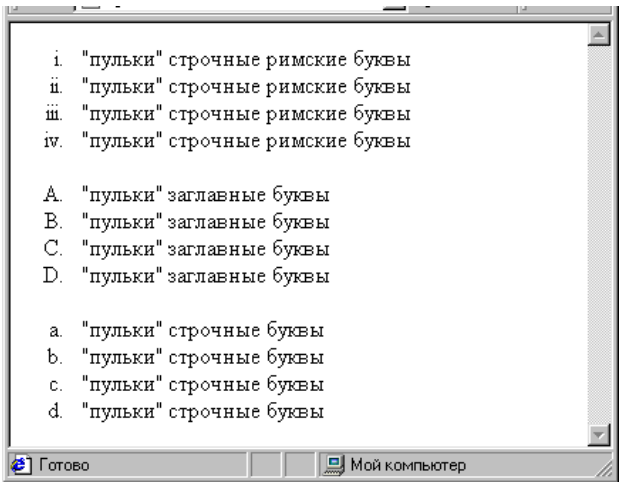

Рис. 59

CSS позволяют вообще отказаться от "пулек". Для этого нужно указать значение атрибута `list-style-type` равным `none`.

Если стандартные формы "пулек" автора страницы не устраивают, он может использовать нестандартные. Для этого ему придется "пульку" нарисовать самому и в виде графического файла разместить на Web-узле. У такой "пульки" есть URL, который используется в CSS для обращения к ней.

```
<UL STYLE="list-style-image:url(bimage.gif);"
> <LI>Элемент списка расположен за чертой
</UL>
```

3.13. ПОЗИЦИОНИРОВАНИЕ

До появления спецификации CSS-P, которая вошла в спецификацию CSS2, размещение блочных элементов HTML-разметки в рабочей области браузера с точностью до пикселя делать на HTML-страницах было нельзя. Единственным средством относительно точного позиционирования были таблицы. Они позволяли точно расположить компоненты HTML-страницы относительно друг друга на плоскости. CSS-P позволяет точно разместить элемент разметки не только относительно других компонентов страницы, но и относительно границ страницы.

Кроме того CSS-P добавляет странице еще одно измерение – элементы разметки могут "наезжать" друг на друга. При этом можно менять порядок "наезда" – перекладывать слои, а также слои можно проявлять (рис. 60).

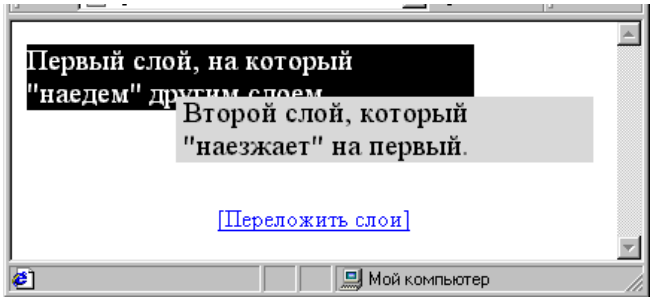

Рис. 60

Термин "слой" вместо "блочный элемент разметки" используется здесь по той причине, что он лучше отражает эффект, который достигается за счет позиционирования.

3.13.1. Координаты и размеры

Стандарт CSS-P позволяет с точностью до пикселя разместить блочный элемент разметки в рабочем поле окна браузера.

CSS-P определяет две системы координат: относительную и абсолютную. Это позволяет обеспечить гибкость размещения элементов как относительно границ рабочего поля окна браузера, так и относительно друг друга.

Блоки – это не абстрактные точки, которые не занимают на плоскости страницы места. Блоки представляют собой прямоугольники, которые "занимают" площадь. Текст и другие компоненты страницы под блоком становятся недоступны пользователю, поэтому линейные размеры блока имеют для создания HTML-страниц не меньшее значение, чем его координаты.

При использовании *абсолютных координат* точка отсчета помещается в верхний левый угол окна браузера, а оси X и Y направлены вправо по горизонтали и вниз по вертикали, соответственно.

Если в этой системе координат некоторый блочный элемент должен быть размещен на `10 px` ниже верхнего обреза рабочей области браузера и на `20 px` правее левого края рабочей области браузера, то его описание будет выглядеть следующим образом:

.example { position:absolute;top:10px; left:20px; }

В данной записи тип системы координат задан атрибутом `position` (значение – `absolute`), координата X задана атрибутом `left` (значение – `20 px`), координата Y – атрибутом `top` (значение – `10 px`).

Атрибуты `top` и `left` определяют координаты верхнего левого угла блока в абсолютной системе координат.

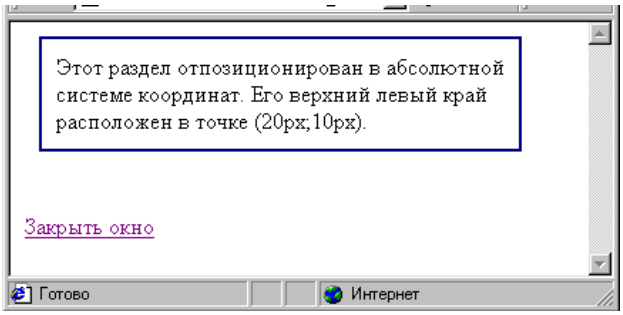

Рис. 61

Значения координат могут быть и отрицательными. Для того, чтобы убрать из отображаемой области блок с линейными размерами `100 px` (высота) на `200 px` (ширина), достаточно позиционировать его следующим образом (рис. 61):

.example {position:absolute; top:-100px;left:-200px; width:200px;height:100px; }

Абсолютное позиционирование применяется тогда, когда либо все содержание страницы должно быть доступно без скроллинга ("прокрутки"), либо когда элементы разметки находятся в начале страницы и их взаимное расположение важно с точки зрения дизайна, например, для использования всплывающих меню.

Относительные координаты позволяют разместить блоки на странице в координатах охватывающего их блока. Преимущества такой системы координат очевидны: она позволяет сохранять взаимное расположение элементов разметки при любом размере окна браузера и его настройках по умолчанию.

В качестве точки отсчета в этой системе координат выбрана точка размещения текущего блока по умолчанию. Ось X при этом направлена горизонтально вправо, а ось Y – вертикально вниз.

Чтобы задать координаты блока, в этой системе применяют запись типа (рис. 62):

```
<DIV STYLE="border-width:1px; border-style:solid;width:100%;height:100px;">
<DIV STYLE="position:relative;top:0px; left:0px;border-width:1px;">
Этот блок находится в точке отсчета относительных координат
</DIV>
<DIV STYLE="position:relative;top:0px; left:50px;border-width:1px;">
А этот блок смещен вправо на 50px
</DIV>
</DIV>
```

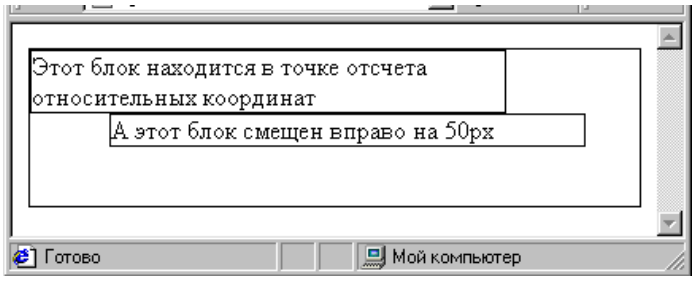

Рис. 62

Для работы с относительной системой координат лучше пользоваться универсальными блоками `DIV`. Это связано с тем, что в Netscape Navigator, например, параграф не может содержать параграфов. Любой блок немедленно закрывает параграф, следовательно, вложить в него что-либо нельзя.

В относительной системе координат можно пользоваться отрицательными смещениями:

```
<DIV STYLE="position:relative; top:0;left:50;border-width:1px; border-style:solid;width:200px;">
<A HREF="javascript: if(flag==0)
{ window.document.layers[2].left=-50;flag=1; }
else
{ window.document.layers[2].left=50;flag=0; };
void(0);">
Сдвинуть слой
</A>
</DIV>
```

В данном примере слой, первоначально сдвинутый на 50 пикселей вправо, после нажатия на гипертекстовую ссылку смещается на 100 пикселей влево, получая отрицательную величину смещения по оси X (*`left`*`:-50 px`). После повторного нажатия на ссылку положение блока восстанавливается.

3.13.2. Линейные размеры блока

Линейные размеры блока задаются двумя параметрами: шириной (`width`) и высотой (`height`) блока. В браузерах ширина и высота блока интерпретируется по-разному.

В Netscape Navigator ширина и высота блока – рекомендуемые параметры. Если текст выходит за эти ограничения, то блок увеличивается до необходимых размеров, а если текста нет вообще, то блок сжимается до маленького квадрата:

```
<P STYLE="width:200px;height:100px;background-color:black;color:white;">
...
</P>
```

Приведенного в примере описания достаточно для получения результата, но в Netscape Navigator для такого блока нужно применить некоторые дополнительные атрибуты (рис. 63):

```
<P STYLE="width:200px;height:100px; background-color:black;color:white;
border-width:1px;border-color:white;">
<SPAN STYLE="color:white;">
...
</SPAN>
</P>
```

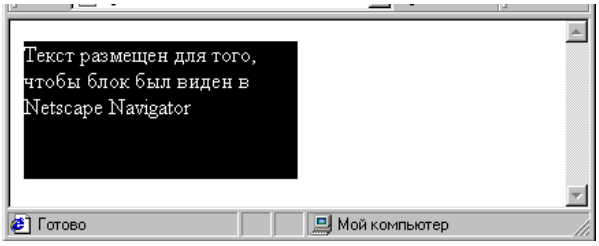

Рис. 63

Без границы блок не будет залит черным цветом, а без *`span`* текст будет отображаться цветом данной страницы по умолчанию. Никакому разумному объяснению такое поведение браузера не поддается, поэтому строить дизайн страниц на этих атрибутах не стоит.

3.13.3. Управление видимостью

Одним из наиболее интересных атрибутов CSS-P является `visibility`. Он позволяет "проявлять" или прятать информацию на HTML-странице.

Например, страница содержит набор кадров, которые можно перелистывать. Для этого просто нужно нажимать на слово "далее" в конце страницы.

Перелистывание реализовано как изменение атрибута `visibility` в JavaScript-функции:

```
function change()
{
next=current+1; if(next>1) next=0;
window.document.all.item("list",
   current).visibility="hidden";
window.document.all.item("list",
   next).visibility="visible";
current=next; }
```

Последовательность "list" – это значение атрибута ID раздела.

Вызов функции при этом помещен в гипертекстовую ссылку со схемой URL "javascript":

```
<A HREF="javascript:change();void(0);">
далее...
</A>
```

Данный атрибут записывается в CSS следующим образом:

```
<DIV STYLE="position:absolute; top:230px;left:55px;width:550px; visibility:hidden;">...</DIV>
```

При программировании атрибута `visibility` следует принимать в расчет тип браузера.

3.13.4. Порядок наложения и область видимости

Абсолютное и относительное позиционирование блоков позволяет придать странице третье измерение. Координаты блока, а точнее левого верхнего угла прямоугольника блока, и его линейные размеры определяют местоположение блока на плоскости. Поскольку блок не является абстрактной точкой, прямоугольники блоков перекрываются, причем в определенном порядке, что позволяет говорить об их размещении в пространстве. Порядок наложения (перекрытия) блоков определяется атрибутом *z-index*.

Вообще, при позиционировании блоков и их наложении друг на друга возникают видимые и невидимые области блоков. CSS позволяет управлять видимостью прямоугольной части блока. За это в спецификации CSS-P отвечает атрибут *clip*.

Если быть более точным, то блоки – это прямоугольные карточки, которые лежат на поверхности, перекрывая друг друга. Расстояние между ними задать нельзя, т.е. полноценного третьего измерения нет.

Порядок перекрытия блоков (или слоев) определяется атрибутом *z-index* (рис. 64). Чем больше значение *z-index*, тем ближе к наблюдателю находится слой.

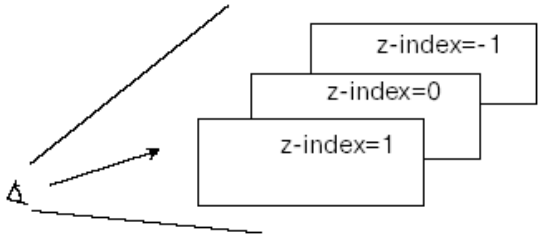

Рис. 64

Как мы видим, *z-index* может принимать и отрицательные значения.

При работе с этим атрибутом следует иметь в виду, что Netscape Navigator и Internet Explorer применяют разные модели описания наложения. В Internet Explorer это просто числовой параметр, который закреплен за блоком и не влияет на значения данного параметра у других блоков. В Netscape Navigator изменение параметра *z-index* одного блока влияет на значения этого параметра у других блоков. Они, словно карточки на столе, перекладываются и получают новые номера.

Совершенно необязательно показывать весь блок целиком. Можно показать только часть блока. Управляется такое отображение параметром *clip* описания CSS:

```
<DIV STYLE="position:absolute;top:0;left:0; width:300;height:50;
clip:rect(0,100,20,0);">
...
</DIV>
```

При вырезании области видимости в Internet Explorer следует учитывать, что работает *clip* только при абсолютном позиционировании. В Netscape Navigator поддерживается как абсолютное позиционирование, так и относительное.

Контрольные вопросы

1. Назовите основные компоненты страницы.
2. Что позволяет определить каскадные таблицы стилей?
3. Назовите способы применения CSS.
4. Каким образом происходит переопределение стиля внутри элемента разметки?

5. Каким образом происходит переопределение стиля в заголовке документа?
6. Как и для каких целей создается внешнее описание стиля?
7. Что означает наследование стилей? Все ли атрибуты стилей могут наследоваться?
8. Каковы правила синтаксиса CSS?
9. Что такое правило?
10. Что такое селектор? Класс?
11. Дайте характеристику блочным и строковым элементам страницы. Какими свойствами обладают блочные элементы?
12. Что такое набивка? Какие стили используются при создании отступов?
13. С помощью каких стилей происходит управление цветом?
14. С помощью каких стилей происходит управление шрифтом?

4. JAVASCRIPT

4.1. ПОНЯТИЕ ОБЪЕКТНОЙ МОДЕЛИ ДОКУМЕНТА

При генерации страниц в Web возникает дилемма, связанная с архитектурой "клиент-сервер". Страницы можно генерировать как на стороне клиента, так и на стороне сервера. В 1995 году специалисты компании Netscape создали механизм управления страницами на клиентской стороне, разработав язык программирования *JavaScript*.

Таким образом, JavaScript – это язык управления сценариями просмотра гипертекстовых страниц Web на стороне клиента.

Основная идея JavaScript состоит в возможности изменения значений атрибутов HTML-тегов и свойств среды отображения в процессе просмотра HTML-страницы пользователем. При этом перезагрузки страницы не происходит.

На практике это выражается в том, что можно, например, изменить цвет фона страницы или интегрированную в документ картинку, открыть новое окно или выдать предупреждение.

Для создания механизма управления страницами на клиентской стороне было предложено использовать объектную модель документа. Суть модели в том, что каждый HTML-тег – это *объект*, который характеризуется тройкой:

- свойства;
- методы;
- события.

Объектную модель можно представить как способ связи между страницами и браузером. Объектная модель – это представление *объектов*, *методов*, *свойств* и *событий*, которые присутствуют и происходят в программном обеспечении браузера в виде, удобном для работы с ними кода HTML и исходного текста сценария на странице. Мы можем с ее помощью сообщать наши пожелания браузеру и далее – посетителю страницы. Браузер выполнит наши команды и соответственно изменит страницу на экране.

Объекты с одинаковым набором *свойств*, *методов* и событий объединяются в классы однотипных *объектов*. Классы – это описания возможных *объектов*. Сами *объекты* появляются только после загрузки документа браузером или как результат работы программы. Об этом нужно всегда помнить, чтобы не обратиться к *объекту*, которого нет.

4.1.1. Свойства

Многие HTML-теги имеют атрибуты. Например, тег якоря `<A ...>...</A>` имеет атрибут `HREF`, который превращает его в гипертекстовую ссылку:

<A HREF=index.htm>Главная</A>

Если рассматривать тег якоря `<A ...>...</A>` как *объект*, то атрибут `HREF` будет задавать *свойство объекта* "якорь". Программист может изменить значение атрибута и, следовательно, *свойство объекта*:

```
document.links[0].href="index.htm";
```

Не у всех атрибутов можно изменять значения. Например, высота и ширина графической картинки определяются по первой загруженной в момент отображения страницы картинке. Все последующие картинки, которые заменяют начальную, масштабируются до нее.

Для общности картины *свойствами* в JavaScript наделены *объекты*, которые не имеют аналогов в HTML-разметке. Например, среда исполнения, называемая *объектом* `Navigator`, или окно браузера, которое является вообще самым старшим *объектом JavaScript*.

4.1.2. Методы

В терминологии JavaScript *методы объекта* определяют функции изменения его *свойств*. Например, с *объектом* "документ" связаны *методы* `open()`, `write()`, `close()`. Эти *методы* позволяют сгенерировать или изменить содержание документа. Приведем простой пример:

```
function hello()
{ id=window.open("","example","width=400, height=150");
  id.focus(); id.document.open();
id.document.write("<H1>Привет!</H1>");
id.document.write("<HR><FORM>");
id.document.write("<INPUT TYPE=button VALUE='Закрыть окно' ");
id.document.write("onClick='window.opener.focus();window.close();'>");
id.document.close();
```

}

В этом примере *метод* `open()` открывает поток записи в документ, *метод* `write()` осуществляет эту запись, *метод* `close()` закрывает поток записи в документ. Все происходит так же, как и при записи в обычный файл. Если у окна есть поле статуса (обычно в нем отображается уровень загрузки документа), то при незакрытом потоке записи в документ в нем будет "метаться" прямоугольник продолжения записи, как это происходит при загрузке документа.

4.1.3. События

Кроме *методов* и *свойств* *объекты* характеризуются *событиями*. Собственно, суть программирования на JavaScript заключается в написании обработчиков этих событий. Например, с *объектом* типа button (тег INPUT типа button – "Кнопка") может происходить событие click, т.е. пользователь может нажать на кнопку. Для этого атрибуты тега INPUT расширены атрибутом обработки события click – onClick. В качестве значения этого атрибута указывается программа обработки события, которую должен написать на JavaScript автор HTML-документа:

<INPUT TYPE=button VALUE="Нажать" onClick="window.alert ('Пожалуйста, нажмите еще раз');">

Обработчики событий указываются в тех тегах, с которыми эти события связаны. Например, тег `BODY` определяет *свойства* всего документа, поэтому обработчик события завершения загрузки всего документа указывается в этом теге как значение атрибута `onLoad`.

4.2. РАЗМЕЩЕНИЕ КОДА НА HTML-СТРАНИЦЕ

В общем случае можно выделить четыре способа функционального применения JavaScript:

- гипертекстовая ссылка (схема URL);
- обработчик события (handler);
- подстановка (entity) (в Microsoft Internet Explorer реализована в версиях от 5.X и выше);
- вставка (тег SCRIPT).

Схема URL (Uniform Resource Locator) – это один из основных элементов Web-технологии. Каждый информационный ресурс в Web имеет свой уникальный URL. URL указывают в атрибуте *`href`* тега `A`, в атрибуте *`src`* тега `IMG`, в атрибуте *`action`* тега `FORM` и т.п.

Основной задачей языка программирования гипертекстовой системы является программирование гипертекстовых переходов. Это означает, что при выборе той или иной гипертекстовой ссылки вызывается программа реализации гипертекстового перехода. В Web-технологии стандартной программой является программа загрузки страницы. JavaScript позволяет поменять стандартную программу на программу пользователя. Для того чтобы отличить стандартный переход по протоколу HTTP от перехода, программируемого на JavaScript, разработчики языка ввели новую схему URL – JavaScript:

<A HREF="JavaScript:JavaScript_код">...</A>

<IMG SRC="JavaScript:JavaScript_код>

В данном случае текст "JavaScript_код" обозначает программы-обработчики на JavaScript, которые вызываются при выборе гипертекстовой ссылки в первом случае и при загрузке картинки – во втором.

Например, при нажатии на гипертекстовую ссылку **Внимание!!!** можно получить окно предупреждения (рис. 65):

<A HREF="JavaScript:alert('Внимание!!!');"> Внимание!!!</A>

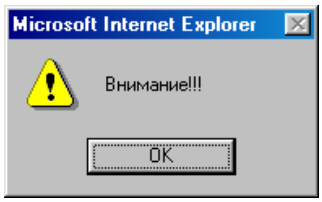

Рис. 65

А при нажатии на кнопку типа submit в форме можно заполнить текстовое поле этой же формы:

```
<FORM NAME=f METHOD=post
ACTION="JavaScript:window.document.f.i.VALUE='Нажали кнопку Click'; void(0);">
<TABLE BORDER=0>
<TR>
<TD><INPUT NAME=i></TD>
<TD><INPUT TYPE=submit VALUE=Click></TD>
<TD><INPUT TYPE=reset VALUE=Reset></TD>
</TABLE>
</FORM>
```

В URL можно размещать сложные программы и вызовы функций. Следует только помнить, что схема JavaScript работает не во всех браузерах, а только в версиях Netscape Navigator и Internet Explorer, начиная с четвертой.

Обработчики событий. Такие программы, как обработчики событий (handler), указываются в атрибутах тегов, с которыми эти события связаны. Например, при нажатии на кнопку происходит событие click:

<FORM><INPUT TYPE=button VALUE="Кнопка" onClick= "window.alert('Внимание');"></FORM>

Подстановки. Подстановка (entity) встречается на Web-страницах довольно редко. Тем не менее, это достаточно мощ-

ный инструмент генерации HTML-страницы на стороне браузера. Подстановки используются в качестве значений атрибутов HTML-тегов. Например, как значение по умолчанию поля формы, определяющего домашнюю страницу пользователя, будет указан URL текущей страницы:

```
<SCRIPT>
function l()
{
  str = window.location.href;
  return(str.length);
}
</SCRIPT>
<FORM><INPUT VALUE="&{window.location.href};" SIZE="&{l()};">
</FORM>
<SCRIPT>
<!-- Это комментарий ...JavaScript-код...// -->
</SCRIPT>
<BODY>
... Тело документа ...
</BODY>
</HTML>
```

Очевидно, что размещать в заголовке документа генерацию текста страницы бессмысленно – он не будет отображен браузером. Поэтому в заголовок помещают декларации общих переменных и функций, которые будут затем использоваться в теле документа. При этом браузер Netscape Navigator более требовательный, чем Internet Explorer. Если не разместить описание функции в заголовке, то при ее вызове в теле документа можно получить сообщение о том, что данная функция не определена.

Приведем пример размещения и использования функции:

```
<HTML>
<HEAD>
<SCRIPT>
function time_scroll()
{
  d = new Date();
  window.status = d.getHours()+":"+d.getMinutes()+":"+d.getSeconds();
  setTimeout('time_scroll();',500);
}
</SCRIPT>
</HEAD>
<BODY onLoad=time_scroll()>
<CENTER>
<H1>Часы в строке статуса</H1>
```

В Internet Explorer 4.0 подстановки не поддерживаются, поэтому пользоваться ими следует аккуратно. Прежде чем выдать браузеру страницу с подстановками, нужно проверить тип этого браузера.

Вставка. Сценарий на JavaScript включается в документ с помощью тега <script>. Помещается между тегами заголовка:

```
<head>
<script>
<!--
  …
  операторы
  сценария
  …
-->
</script>
</head>
```

Помещение сценария в раздел head документа приводит к тому, что сценарий будет загружен до того, как потребуется его выполнить.

Код сценария заключается в теги комментария html для того, чтобы старые браузеры, не понимающие JavaScript, не отображали этот код на экране.

Регистр, в котором написаны буквы, в JavaScript имеет значение.

Сам тег <script> включает атрибут *language*, который определяет язык и принимает следующие значения:

```
language = "javascript"
   = "VBscript"
   = "JAVASCRIPTcript"
   = "TCL"
```

По умолчанию браузер интерпретирует сценарий с языка JavaScript, поэтому атрибут language можно опустить.

Текст сценария может храниться в файле. Тогда в теге <script> должен быть атрибут *src="URL-адрес"* с ссылкой на этот файл.

Файлы с текстами программ на JavaScript имеют расширение JavaScript.

Элемент script может располагаться как внутри секции head, так и внутри body.

Большой Web-документ часто содержит несколько сценариев JavaScript.

Тег <SCRIPT>, введенный в элементе <HEAD> документа HTML всегда выполняется первым. Хотя этот сценарий и не позволяет отображать на экране данные, он прекрасно подходит для определения функций.

Сценарий, добавленный в тело документа HTML, выполняется всегда после сценариев, введенных в его заголовке. Несколько сценариев в теле документа HTML выполняются в порядке их следования.

Обработчики событий запускаются при выполнении соответствующих событий. Например, обработчик onLoad выполняется сразу после загрузки страницы. Чтобы правильно задать функцию, используемую в обработчике событий, старайтесь всегда определять ее в заголовке документа HTML.

4.3. ИЕРАРХИЯ КЛАССОВ

4.3.1. Объекты JavaScript

Объектно-ориентированный язык программирования предполагает наличие иерархии классов *объектов* (рис. 66). В JavaScript такая иерархия начинается с класса *объектов* `Window`, т.е. каждый *объект* приписан к тому или иному окну. Для обращения к любому *объекту* или его *свойству* указывают полное или частичное имя этого *объекта* или *свойства объекта*, начиная с имени *объекта* старшего в иерархии, в который входит данный *объект*.

Объект *navigator* относится к самому браузеру и его свойства позволяют определить характеристики программы просмотра:

- appName – содержит имя браузера (например, "Microsoft Internet Explorer");
- appVersion – содержит информацию о версии браузера (например, "4.0 (Compatible; MSIE 4.01; Windows 95)").

Каждая страница, в дополнение к объекту *navigator*, обязательно имеет еще 5 объектов:

- *window* – объект верхнего уровня, свойства которого применяются ко всему окну, в котором отображается документ;

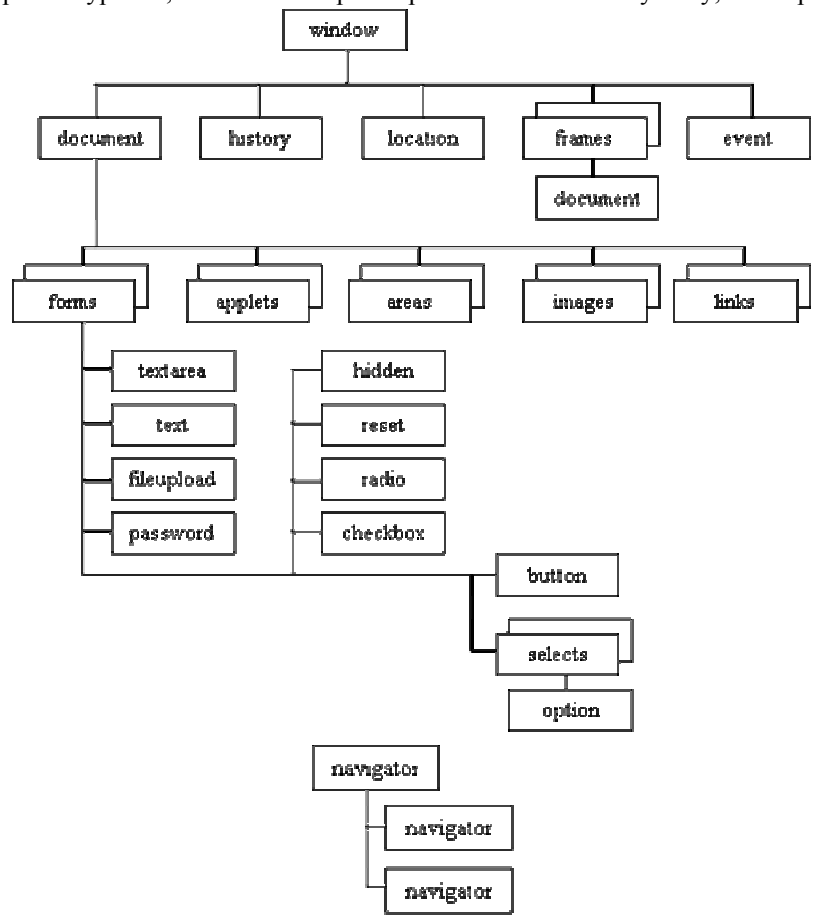

Рис. 66

- *document*. Его свойства определяются содержимым самого документа: связи, цвет фона, формы и т.д.;
- *location*. Его свойства связаны с URL адресом отображаемого документа;
- *history*. Представляет адреса ранее загрузившихся html страниц;
- *event*. Предоставляет доступ к свойствам происшедшего события.

Например, рассмотрим следующую форму:
<form name="form1">
фамилия: <input type="text" name="StudentName" size=20>
курс: <input type="text" name="course" size=2>
</form>
Пример обращения к фамилии:
document.form1.StudentName.value

4.3.2. Свойства и методы ключевых объектов

1. Window (окно). Создается автоматически при запуске браузера.

Методы:

- open () – создание окна;
- close () – закрытие окна.

2. Document. Содержит информацию о загруженной странице. Для каждой страницы создается один объект document. Некоторые его свойства соответствуют параметрам тега body. Свойство URL содержит адрес загруженного документа.

Методы:

write, writeln записывают в документ информацию и тем самым позволяют динамически его создавать.

3. Location. Связан с текущим URL адресом. Его свойства позволяют получать информацию о host-машине, с которой в данный момент связан браузер.

Свойства:

- hostname – содержит имя хоста;
- port – номер порта, к которому подсоединен браузер на хост-машине.

Методы:

- reload () – перезагружает в браузер текущую страницу;
- replace () – загружает в окно браузера страницу, адрес которой задан в качестве его параметра.

4. History. Содержит список адресов html документов, ранее загруженных в браузер.

Свойства:

- current – текущая страница;
- next – следующая страница;
- previous – предыдущая страница.

Метод:

- go () – загружает страницу из списка посещенных.

Текущая страница имеет индекс 0, предыдущая индексируется отрицательными целыми числами, последующая – положительным целым числом.

5. Form. Порождается объектом document и сам порождает подчиненные объекты. Ссылка на этот объект осуществляется с помощью переменной, определенной в параметре name тега form.

В документе может быть несколько форм, поэтому введено свойство-массив *forms*, в котором содержатся ссылки на все формы документа.

П р и м е р обращения:

По номеру:
document.forms[0]
document.forms[1]
По имени:
document.forms["form1"]

Если в документе одна форма (с именем form1):
document.form1.

Все элементы формы порождают соответствующие объекты, подчиненные объекту родительской формы:
<input type="text" name="txt1">.

Варианты обращения к полю txt1:
document.forms[0].txt1;
document.forms["form1"].txt1;
document.forms.txt1.

Каждый объект form имеет также свойство-массив *elements*, содержащий ссылки на все подчиненные форме элементы в том порядке, в котором они определены в документе html.

Объекты формы имеют свойство *name*, значение которого равно значению параметра *name* тега *input*, а также свойство *value*, значение которого определяется смыслом параметра *value* элемента формы.

6. Объект frame.

Страница с заданным количеством фреймов образует иерархическую модель объектов *frame*.

На верхнем уровне объект *top* – родитель всех фреймов на странице.

Для ссылки на фреймы страницы можно использовать либо символические имена, либо свойство-массив *frames* объекта *top*, в котором содержатся ссылки на все фреймы страницы.

Свойство *location* объекта *frame* содержит адрес загруженного во фрейм документа.

Изменение этого значения приведет к загрузке нового документа в соответствующий фрейм.

<input type="button" value="Пример1" onClick="JavaScript:top. ContentFrame.location='Пример1.html'">

Если загружаемый во фрейм документ сам содержит набор фреймов, то все фреймы, отображенные в заданном фрейме, являются его подчиненными.

7. Свойства-массивы объектов.

Некоторые объекты имеют свойства, которые являются массивами. Они используются для хранения информации о подчиненных объектах, когда их количество заранее неизвестно.

Объект	Свойство	Описание
Document	Anchors	<A>
	Applets	<APPLET>
	Forms	<FORM>
	Images	<IMG>
	Links	<AREA HREF=<...> <A HREF=<...>
Function	Arguments	Отражает параметры функции
Forms	Elements	<FORM>
Select	Options	Объект select, теги <option>
Window	Frames	теги <frame> в <frameset>
History	Historys	Отражает элементы объекта history

8. Объект event. Каждое событие порождает ассоциированный с ним объект *event*. Этот объект содержит всю информацию о событии, и ее можно передать процедуре обработки события. Эта информация зависит от типа события.

Например, event MouseDown содержит:

- информацию о типе события (свойство type);
- какая кнопка мыши была нажата (which);
- координаты курсора (screenx, screeny).

Совместно с обработчиками событий объекты event позволяют производить достаточно тонкую обработку событий.

Процедуру обработки события можно вызвать двумя способами:

- Явно, назначив ссылку на процедуру обработки события в соответствующем свойстве объекта;
- Неявно – в параметре обработки события тега соответствующего элемента.

П р и м е р :

```
<form name="form1">
<input type="button" name="button1" value="Узнай событие">
<script>
  document.form1.button1.onMouseDown=showEventType
</script>
</form>
function showEventType(e){
  alert ("Произошло событие: "+e.type)
}
```

В объявлении функции *showEventType* присутствует параметр *e*, свойство *type* которого выводится в диалоговом окне.

При явном вызове процедуры обработки события объект *event* передается ей по умолчанию, поэтому в данном случае печатается значение свойства *type* объекта *event*, т.е. тип события *MouseDown*.

При неявном вызове требуется задание обращения к процедуре в параметре *onMouseDown* тега *input*, при этом необходимо явно указывать параметр *event*.

П р и м е р :

```
<form name="form1">
<input type="button" name="button1" value="Узнай событие" onMouseDown="showEventType(event)">
</form>
```

4.4. ПРОГРАММИРОВАНИЕ СВОЙСТВ ОКНА БРАУЗЕРА

Класс объектов `Window` – это самый старший класс в иерархии объектов JavaScript. К нему относятся объект `Window` и объект `Frame`. Объект `Window` ассоциируется с окном программы-браузера, а объект `Frame` – с окнами внутри окна браузера, которые порождаются последним при использовании автором HTML-страниц тегов `FRAMESET` и `FRAME`.

При программировании на JavaScript чаще всего используют следующие свойства и методы объектов типа `Window`:

Свойства	Методы	События
Status Location History Navigator	Open() Close() Focus()	Событий нет

Объект Window создается только в момент открытия окна. Все остальные объекты, которые порождаются при загрузке страницы в *окно*, есть свойства объекта Window. Таким образом, у Window могут быть разные свойства при загрузке разных страниц.

Поле статуса. Поле статуса – это первое, что начали использовать авторы HTML-страниц из арсенала JavaScript. Калькуляторы, игры, математические вычисления и другие элементы выглядели слишком искусственно. На их фоне бегущая строка в поле статуса была изюминкой, которая могла действительно привлечь внимание пользователей к Web-узлу. Постепенно ее популярность сошла на нет. Бегущие строки стали редкостью, но программирование поля статуса встречается на многих Web-узлах.

Полем статуса (status bar) называют среднее поле нижней части окна браузера сразу под областью отображения HTML-страницы. В поле статуса отображается информация о состоянии браузера (загрузка документа, загрузка графики, завершение загрузки, запуск апплета и т.п.). Программа на JavaScript имеет возможность работать с этим полем как с изменяемым свойством окна. При этом фактически с ним связаны два разных свойства:

- window.status;
- window.defaultStatus.

Разница между ними заключается в том, что браузер на самом деле имеет несколько состояний, связанных с некоторыми событиями. Состояние браузера отражается в сообщении в поле статуса. По большому счету, существуют только два состояния: нет никаких событий (defaultStatus) и происходят какие-то события (status).

Свойство status связано с отображением сообщений о событиях, отличных от простой загрузки страницы. Например, когда курсор мыши проходит над гипертекстовой ссылкой, URL, указанный в атрибуте HREF, отображается в поле статуса. При попадании курсора мыши на поле, свободное от ссылок, в поле статуса восстанавливается сообщение по умолчанию (Document:Done). Эта техника реализована в примере при переходе на описание свойств status и defaultStatus:

```
<A HREF=#status onMouseover="window.status='Jump to status description';return true;"
onMouseout="window.status='Status bar programming';return true;">window.status</A>
```

Свойство defaultStatus определяет текст, отображаемый в поле статуса, когда никаких событий не происходит. Например, определим это свойство при загрузке документа:

```
<BODY onLoad="window.defaultStatus='Status bar programming';">
```

Это сообщение появляется в тот момент, когда загружены все компоненты страницы (текст, графика, апплеты и т.п.). Оно восстанавливается в строке статуса после возврата из любого события, которое может произойти при просмотре документа. Любопытно, что движение мыши по свободному от гипертекстовых ссылок полю страницы приводит к постоянному отображению defaultStatus.

Поле location. В поле location (рис. 67) отображается URL загруженного документа. Если пользователь хочет вручную перейти к какой-либо странице (набрать ее URL), он делает это в поле location. Поле располагается в верхней части окна браузера ниже панели инструментов, но выше панели личных предпочтений.

Location – это объект. Из-за изменений в версиях JavaScript класс Location входит как подкласс и в класс Window, и в класс Document. Мы будем рассматривать Location только как window.location. Кроме того, Location – это еще и подкласс класса URL, к которому относятся также объекты классов Area и Link. Location наследует все свойства URL, что позволяет получить доступ к любой части схемы URL.

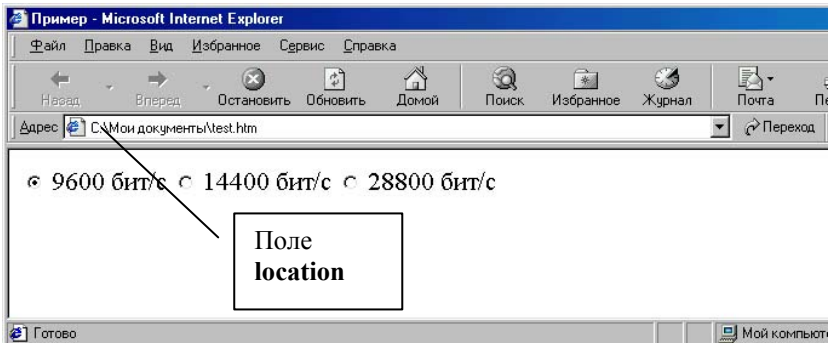

Рис. 67

Рассмотрим характеристики и способы использования объекта Location.

Свойства. Предположим, что браузер отображает страницу, расположенную по адресу http://tstu.ru:80/r/dir/page?search#mark.

Тогда свойства объекта Location примут следующие значения:

window.location.href = http://tstu.ru:80/r/dir/page?search#mark;
window.location.protocol = http;
window.location.hostname = tstu.ru;
window.location.host = tstu.ru:80;
window.location.port = 80
window.location.pathname = /r/dir/;

window.location.search = search;
window.location.hash = mark;

Методы. Методы `Location` предназначены для управления загрузкой и перезагрузкой страницы. Это управление заключается в том, что можно либо перезагрузить документ `(reload)`, либо загрузить `(replace)`. При этом в трассу просмотра страниц (`history`) информация не заносится.

Метод `reload()` полностью моделирует поведение браузера при нажатии на кнопку Reload в панели инструментов. Если вызывать метод без аргумента или указать его равным true, то браузер проверит время последней модификации документа и загрузит его либо из кэша (если документ не был модифицирован), либо с сервера. Такое поведение соответствует простому нажатию на кнопку Reload. Если в качестве аргумента указать false, то браузер перезагрузит текущий документ с сервера, несмотря ни на что. Такое поведение соответствует одновременному нажатию на Reload и кнопку клавиатуры Shift (Reload+Shift).

Метод `replace()` позволяет заменить одну страницу другой таким образом, что это замещение не будет отражено в трассе просмотра HTML-страниц (History), и при нажатии на кнопку Back из панели инструментов пользователь всегда будет попадать на первую загруженную обычным способом (по гипертекстовой ссылке) страницу. Напомним, что при изменении свойств `Location` также происходит перезагрузка страниц, но в этом случае записи об их посещении в `History` пропадают.

История посещений (*History*). История посещений (трасса) страниц World Wide Web позволяет пользователю вернуться к странице, которую он просматривал несколько минут (часов, дней) назад. История посещений в JavaScript трансформируется в объект класса `History`. Этот объект указывает на массив URL-страниц, которые пользователь посещал и которые он может получить, выбрав из меню браузера режим GO. Методы объекта `History` позволяют загружать страницы, используя URL из этого массива.

Чтобы не возникло проблем с безопасностью браузера, путешествовать по History можно, только используя индекс URL. При этом URL, как текстовая строка, программисту недоступен. Чаще всего этот объект используют в примерах или страницах, на которые могут быть ссылки из нескольких разных страниц, предполагая, что можно вернуться к странице, из которой пример будет загружен:

```
<FORM><INPUT TYPE=button VALUE="Назад" onClick=history.back()> </FORM>
```

Данный код отображает кнопку "Назад", нажав на которую мы вернемся на предыдущую страницу.

Тип браузера (*объект Navigator*). В связи с войной браузеров (которая, можно считать, уже закончилась в пользу Microsoft Internet Explorer) стала актуальной задача настройки страницы на конкретную программу просмотра. При этом возможны два варианта: определение типа браузера на стороне сервера и определение типа браузера на стороне клиента. Для последнего варианта в арсенале объектов JavaScript существует объект `Navigator`. Этот объект – свойство объекта `Window`.

Рассмотрим простой пример определения типа программы просмотра:

```
<FORM><INPUT TYPE=button VALUE="Тип навигатора" on-
                    Click="window.alert(window.navigator.userAgent);"></FORM>.
```

При нажатии на кнопку отображается окно предупреждения. В нем содержится строка `userAgent`, которую браузер помещает в соответствующий HTTP-заголовок.

Эту строку можно разобрать по компонентам, например:

```
navigator.appName = Microsoft Internet Explorer;
navigator.appCodeName = Mozilla;
navigator.appVersion = 4.0 (compatible; MSIE 5.5; Windows 98);
navigator.userAgent = Mozilla/4.0 (compatible; MSIE 5.5; Windows 98).
```

У объекта `Navigator` есть еще несколько интересных с точки зрения программирования применений. Например, проверка поддержки Java.

Проиллюстрируем эту возможность на примере:

```
<SCRIPT>
document.write("<P ID=red>");
if(navigator.javaEnabled()==true)
  document.write("Ваша программа поддерживает исполнение Java-апплетов");
if(navigator.javaEnabled()==false)
  document.write("<FONT COLOR=red>Ваша программа не поддерживает исполнение Java-апплетов</FONT>");
</SCRIPT>
</example>
```

Аналогично можно проверить форматы графических файлов, которые поддерживает браузер:

```
<SCRIPT>
if(navigator.mimeTypes['image/gif']!=null)
  document.write("Ваш браузер поддерживает GIF<BR>");
if(navigator.mimeTypes['image/tif']==null)
  document.write("Ваш браузер не поддерживает TIFF");
</SCRIPT>
```

Управление окнами. Что можно сделать с окном? Открыть (создать), закрыть (удалить), положить его поверх всех других открытых окон (передать фокус). Кроме того, можно управлять свойствами окна и свойствами подчиненных ему объектов. Наиболее популярные методы управления окнами:

- alert();
- confirm();
- prompt();
- open();
- close();
- focus();
- setTimeout();
- clearTimeout().

Здесь не указаны только два метода: scroll() и blur().

Первый позволяет прокрутить окно на определенную позицию. Но его очень сложно использовать, не зная координат окна. Последнее является обычным делом, если только не используется технология программирования слоев или CSS (Cascading Style Sheets).

Второй метод уводит фокус с окна. При этом совершенно непонятно, куда этот фокус будет передан. Лучше целенаправленно передать фокус, чем просто его потерять.

***Window.alert*().** Метод alert() позволяет выдать окно предупреждения:

```
<A HREF="javascript:window.alert('Внимание')">
Повторите запрос!</A>
```

***Window.confirm*().** Метод confirm() позволяет задать пользователю вопрос, на который тот может ответить либо положительно, либо отрицательно:

```
<FORM>
<INPUT TYPE=button VALUE="Вы знаете JavaScript?"
onClick="if(window.confirm('Знаю все')==true)
{ document.forms[0].elements[1].value='Да'; }
else {
  document.forms[0].elements[1].value='Нет';
};"><BR>
</FORM>
```

***Window.prompt*().** Метод prompt() позволяет принять от пользователя короткую строку текста, которая набирается в поле ввода информационного окна:

```
<FORM>
<INPUT TYPE=button VALUE="Открыть окно ввода"
onClick="document.forms[1].elements[1].value=window.prompt('Введите сообщение');">
<INPUT SIZE=30>
</FORM>
```

Введенную пользователем строчку можно присвоить любой переменной и потом разбирать ее в JavaScript-программе.

***Window.open*().** У этого метода окна атрибутов больше, чем у некоторых объектов. Метод open() предназначен для создания новых окон. В общем случае его синтаксис выглядит следующим образом:

open("URL","window_name","param,param,...", replace);

где URL – страница, которая будет загружена в новое окно; window_name – имя окна, которое можно использовать в атрибуте TARGET в тегах A и FORM.

Параметры	Назначение
Replace	Позволяет при открытии окна управлять записью в массив History
Param	Список параметров
Width	Ширина окна в пикселах
Height	Высота окна в пикселах
Toolbar	Создает окно с системными кнопками браузера
Location	Создает окно с полем location
Directories	Создает окно с меню предпочтений пользователя
Status	Создает окно с полем статуса status
Menubar	Создает окно с меню
Scrollbar	Создает окно с полосами прокрутки
Resizable	Создает окно, размер которого можно будет изменять

Приведем следующий пример:

```
<FORM>
```

```
<INPUT TYPE=button VALUE="Простое окно" onClick="window.open ('about:blank','test1',
'directories=no,height=200,location=no,menubar=no,resizable=no,scrollbars=no, status=no,
toolbar=no,width=200');">
<INPUT TYPE=button VALUE="Сложное окно" onClick="window. open('about:blank','test2',
'directories=yes, height=200,location=yes,menubar=yes,resizable=yes, scrollbars=yes,
status=yes,toolbar=yes,width=200');">
</FORM>
```

При нажатии кнопки "простое окно" получаем окно со следующими параметрами:

- `directories=no` – окно без меню;
- `height=200` – высота 200 px;
- `location=no` – поле location отсутствует;
- `menubar=no` – без меню;
- `resizable=no` – размер изменять нельзя;
- `scrollbars=no` – полосы прокрутки отсутствуют;
- `status=no` – статусная строка отсутствует;
- `toolbar=no` – системные кнопки браузера отсутствуют;
- `width=200` – ширина 200.

При нажатии кнопки "сложное окно" получаем окно, где

- `directories=yes` – окно с меню;
- `height=200` – ширина 200 px;
- `location=yes` – поле location есть;
- `menubar=yes` – меню есть;
- `resizable=yes` – размер изменять можно;
- `scrollbars=yes` – есть полосы прокрутки;
- `status=yes` – статусная строка есть;
- `toolbar=yes` – системные кнопки браузера есть;
- `width=200` – длина 200.

***Window.close*().** Метод close() – это оборотная сторона медали метода open(). Он позволяет закрыть окно. Чаще всего возникает вопрос, какое из окон, собственно, следует закрыть. Если необходимо закрыть текущее, то:

```
window.close();
self.close();
```

Если необходимо закрыть родительское окно, т.е. окно, из которого было открыто текущее, то:

```
window.opener.close();
```

Если необходимо закрыть произвольное *окно*, то тогда сначала нужно получить его идентификатор:

```
id=window.open();
...
id.close();
```

Как видно из последнего примера, закрывают окно не по имени (значение атрибута `TARGET` тут ни при чем), а используют указатель на объект.

***Window.focus*().** Метод `focus()` применяется для передачи фокуса в окно, с которым он использовался. Передача фокуса полезна как при открытии окна, так и при его закрытии, не говоря уже о случаях, когда нужно выбирать окна. Рассмотрим пример.

Открываем окно и, не закрывая его, снова откроем окно с таким же именем, но с другим текстом. Новое окно не появилось поверх основного окна, так как фокус ему не был передан. Теперь повторим открытие окна, но уже с передачей фокуса:

```
function myfocus(a)
{
id = window.open("","example","scrollbars,width=300,height=200");
//открываем окно и заводим переменную с указателем на него
//если окно с таким именем существует, то новое окно не создается,
//а открывается поток для записи в окно с этим именем
if(a==1)
{
id.document.open();
//открываем поток ввода в уже созданное окно
id.document.write("<CENTER>>Открыли окно в первый раз");
//Пишем в этот поток
}
if(a==2)
{
id.document.open();
id.document.write("<CENTER>Открыли окно во второй раз");
```

```
}
if(a==3)
{
id.focus();
//передаем фокус, затем выполняем те же действия, что и в предыдущем случае
id.document.open();
id.document.write("<CENTER>Открыли окно во второй раз");
}
id.document.write("<FORM><INPUT TYPE=button
onClick='window.close();' VALUE='Закрыть окно'></CENTER>");
id.document.close();
}
```

Поскольку мы пишем содержание нового окна из окна старого (родителя), то в качестве указателя на объект используем значение переменной `id`.

***Window.setTimeout*().** Метод setTimeout() используется для создания нового потока вычислений, исполнение которого откладывается на время (ms), указанное вторым аргументом:

```
idt = setTimeout("JavaScript_код",Time);
```

Типичное применение этой функции – организация автоматического изменения свойств объектов. Например, можно запустить часы в поле формы:

```
var flag=0;
var idp=null;
function myclock()
{
if(flag==1)
{
d = new Date();
window.document.c.f.value = d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds(); }
idp=setTimeout("myclock();",500);
}
function flagss()
{
if(flag==0) flag=1; else flag=0;
}
...
<FORM NAME=c>
Текущее время:<INPUT NAME=f size=8><INPUT TYPE=button VALUE="Start/Stop"
onClick=flagss()>
</FORM>
```

Нужно иметь в виду, что поток порождается всегда, даже в том случае, когда часы стоят. Если бы он создавался только при значении переменной `flag,` равном единице, то при значении 0 он исчез бы, тогда при нажатии на кнопку часы продолжали бы стоять.

Window.clearTimeout. Метод `clearTimeout()` позволяет уничтожить поток, вызванный методом setTimeout(). Очевидно, что его применение позволяет более эффективно распределять ресурсы вычислительной установки. Для того чтобы использовать этот метод в примере с часами, нам нужно модифицировать функции и форму:

```
var idp1 = null;
function start()
{
d = new Date();
window.document.c1.f1.value = d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds();
idp1=setTimeout("start();",500);
}
function stop()
{
clearTimeout(idp1);idp1=null;
}
...
<FORM NAME=c1>
Текущее время:<INPUT NAME=f1 size=8>
<INPUT TYPE=button VALUE="Start" onClick="if(idp1==null)start();">
<INPUT TYPE=button VALUE="Stop" onClick="if(idp1!=null)stop();">
```

```
</FORM>
```

В данном примере для остановки часов используется метод `clearTimeout()`. При этом, чтобы не порождалось множество потоков, проверяется значение указателя на объект потока.

4.5. ФРЕЙМЫ (FRAMES)

Фреймы – это несколько видоизмененные окна. Отличаются они от обычных окон тем, что размещаются внутри них. У фрейма не может быть ни панели инструментов, ни меню, как в обычном окне. В качестве поля статуса фрейм использует поле статуса окна, в котором он размещен. Существует и ряд других отличий.

Иерархия фреймов. Рассмотрим сначала простой пример. Разделим экран на две вертикальные колонки:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET COLS="50%,*">
<FRAME NAME=left SRC=left.html>
<FRAME NAME=right SRC=right.html>
</FRAMESET>
</HTML>
```

Назовем окно, в которое помещают фреймы, `_top`(`_parent`).

Усложним пример: разобьем правый фрейм на два по горизонтали:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET COLS="50%,*">
<FRAME NAME=left SRC=left.html>
<FRAMESET ROWS="50%,*">
<FRAME NAME=top SRC=top.html>
<FRAME NAME=bottom SRC=bottom.html>
</FRAMESET>
</FRAMESET>
</HTML>
```

Обратите внимание на два момента: во-первых, следует различать `_top` и `top`, во-вторых, исчез фрейм `right`. По поводу первого замечания: `_top` – это зарезервированное имя старшего окна, а `top` – имя фрейма, которое назначил ему автор страницы. По поводу второго замечания: старшим окном для всех фреймов является все окно браузера, фрейма с именем `right` в данном случае не существует.

Для того чтобы он появился, нужно свести оба наших примера в один. Это значит, что во фрейм `right` мы снова должны загрузить фреймовый документ.

Первый документ:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET COLS="50%,*">
<FRAME NAME=left SRC=left.html>
<FRAME NAME=right SRC=right.html>
</FRAMESET>
</HTML>
```

Второй документ (right.htm):

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET ROWS="50%,*">
<FRAME NAME=top SRC=top.html>
<FRAME NAME=bottom SRC=bottom.html>
</FRAMESET>
</HTML>
```

В этом случае подчинение страниц будет выглядеть иначе, чем в примере с тремя фреймами.

Таким образом, мы получили тот же результат, что и с тремя фреймами и одним старшим окном. Однако этот вариант более гибкий: он позволяет задействовать фрейм, содержащий горизонтальную разбивку.

Именование фреймов. Обратиться к фрейму можно либо по имени, либо как к элементу массива `frames[]`. Рассмотрим HTML-документ:

```
<HTML>
<HEAD>
...
</HEAD>
<FRAMESET COLS="20%,*">
<FRAME NAME=left SRC=frame1.htm>
<FRAME NAME=right SRC=frame2.htm>
</FRAMESET>
</HTML>
```

Предположим, что на странице, загруженной в правый фрейм, есть две картинки. Для изменения свойства `src` второй из них можно использовать следующие записи:

```
top.frames[1].images[1].src="pic.gif";
```

или

```
top.right.images[1].src="pic.gif";
```

В связи с индексированием фреймов возникает вопрос о том, как они нумеруются в одномерном встроенном массиве фреймов объекта `Window`. Проиллюстрируем это на примере (рис. 68):

```
<FRAMESET ROWS="50,*,50">
<FRAME NAME=top SRC=top.html>
<FRAMESET COLS="100,*,100">
<FRAME NAME=left SRC=left.html>
<FRAME NAME=center SRC=center.html>
<FRAME NAME=right SRC=right.html>
</FRAMESET>
<FRAME NAME=bottom SRC=bottom.html>
</FRAMESET>
```

Рис. 68

Построим теперь столбец из трех фреймов (рис. 69):

```
<FRAMESET COLS="100,*,100">
<FRAME NAME=left SRC=top.html>
<FRAMESET ROWS="60,*,60">
<FRAME NAME=top SRC=left.html>
<FRAME NAME=center SRC=center.html>
<FRAME NAME=bottom SRC=right.html>
</FRAMESET>
<FRAME NAME=right SRC=bottom.html>
</FRAMESET>
```

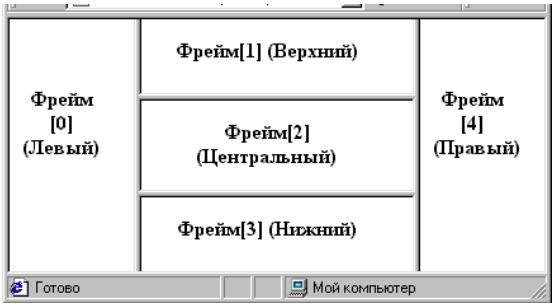

Рис. 69

Таким образом, при нумеровании фреймов в одномерном массиве фреймов на странице система придерживается правила "слева направо, сверху вниз". Вкладывая наши примеры друг в друга, можно получить правильную индексацию страниц при любой сложной фреймовой структуре.

Передача фокуса во фрейм. Обычной задачей при разработке типового Web-узла является загрузка результатов исполнения CGI-скрипта во фрейм, отличный от фрейма, в котором вводятся данные для этого скрипта. Если путь загрузки результатов фиксированный, то можно просто использовать атрибут *target* формы. Сложнее, если результат работы должен быть загружен в разные фреймы, в зависимости от выбранной кнопки, например.

Эту задачу можно решать по-разному: открывать ранее открытое окно или переназначать свойство *target*. Последнее решение, конечно, более изящное, с него и начнем:

```
function load()
{
 if(self.document.f.s.options[document.f.s.selectedIndex].text=="top")
 {
  document.f.target = "mytop";
  self.top.frames[2].document.open();
  self.top.frames[2].document.close();
 }
 else
 {
  document.f.target = "mybottom";
  self.top.frames[1].document.open();
  self.top.frames[1].document.close();
 }
 return true;
}
```

Функция `load()` вызывается как обработчик события submit, она является логической функцией. Возврат значения `true` позволяет реализовать перезагрузку документа.

Теперь рассмотрим второй вариант. Его идея состоит в том, что при попытке открыть окно с именем существующего окна новое окно не открывается, а используется уже открытое. Фрейм – это тоже окно, поэтому на него данное правило распространяется, но вот функция, которая реализует этот вариант, отличается от предыдущей:

```
function load()
{
 if(self.document.f.s.options[document.f.s.selectedIndex].text=="top")
 {
  window.open("./framer.htm","mytop");
  self.top.frames[2].document.open();
  self.top.frames[2].document.close();
 }
 else
 {
  window.open("./framer.htm","mybottom");
  self.top.frames[1].document.open();
  self.top.frames[1].document.close();
 }
 return false;
}
```

4.6. ПРОГРАММИРОВАНИЕ ФОРМ

Если рассматривать программирование на JavaScript в исторической перспективе, то первыми объектами, для которых были разработаны методы и свойства, стали поля форм. Обычно тег `FORM` и поля форм именованы:

```
<FORM NAME=f_name METHOD=post
```

```
ACTION="javascript:void(0);">
<INPUT NAME=i_name SIZE=30 MAXLENGTH=30>
</FORM>
```

Поэтому в программах на JavaScript к ним обращаются по имени:

```
window.document.f_name.i_name.value="Текстовое поле";
```

Того же эффекта можно достичь, используя массив форм загруженного документа:

```
window.document.forms[0].elements[0].value="Текстовое поле";
```

В данном примере не только к форме, но и к полю формы мы обращаемся как к элементу массива.

Рассмотрим подробнее объект *Form*, который соответствует тегу `FORM`.

Свойства	Методы	События
Action Method Target Elements[] Encoding	Reset() Submit()	OnReset OnSubmit

Сами по себе методы, свойства и события объекта Form используются нечасто. Их переопределение обычно связано с реакцией на изменения значений полей формы.

Свойство `action` отвечает за вызов скрипта (CGI-скрипта). В нем указывается его (скрипта) URL. Но там, где можно указать URL, можно указать и его схему JavaScript:

```
<FORM METHOD=post
    ACTION="javascript:window.alert('We use JavaScript-code as an URL'); void(0);">
<INPUT TYPE=submit VALUE="Продемонстрировать JavaScript в action">
</FORM>
```

Применение `void(0)` отменяет перезагрузку документа, и браузер не генерирует событие submit, т.е. не обращается к серверу при нажатии на кнопку, как при стандартной обработке форм.

Свойство `method` определяет метод доступа к ресурсам HTTP-сервера из программы-браузера. В зависимости от того, как автор HTML-страницы собирается получать и обрабатывать данные из формы, он может выбрать тот или иной метод доступа. На практике чаще всего используются методы `GET` и `POST`.

JavaScript-программа может изменить значение этого свойства:

```
<FORM NAME=m ACTION="javascript:window.alert('Мы используем JavaScript-код в качестве an URL');void(0);">
<SCRIPT>
document.write("<FONT COLOR=navy>По умолчанию установлен метод</FONT>"+document.m.method+".");
</SCRIPT>
<INPUT TYPE=button onClick="window.document.main.document. m.method='post';" VALUE="Метод POST">
<INPUT TYPE=button onClick="window.document.main.document. m.method='get';" VALUE="Метод GET">
<INPUT TYPE=submit VALUE="JavaScript в ACTION">
</FORM>
```

По умолчанию установлен метод `GET`.

В данном примере стоит обратить внимание на два момента:

1. Прежде чем открывать окно предупреждения, следует нажать кнопку "Метод POST". Если этого не сделать, то появится сообщение об ошибке JavaScript. Формирование URL происходит при генерации события submit, а вызов скрипта – после того, как событие сгенерировано. Поэтому вставить переопределение метода в обработчик события нельзя, так как к этому моменту будет уже сгенерирован URL, который, в свою очередь, будет JavaScript-программой с символом "`?`" на конце. Переопределение метода должно быть выполнено раньше, чем произойдет событие submit.
2. В тело документа через тег `SCRIPT` встроен JavaScript-код, который сообщает метод доступа, установленный в форме по умолчанию. Этот тег расположен сразу за тегом `FORM`. Ставить его перед тегом `FORM` нельзя, так как в момент получения интерпретатором управления объект `FORM` не будет создан, и, следовательно, работать с его свойствами не представляется возможным.

Никаких других особенностей свойство `method` не имеет. В данном свойстве можно указать и другие методы доступа, отличные от `GET` и `POST`, но это требует дополнительной настройки сервера.

Свойство `target` определяет имя окна, в которое следует загружать результат обращения к CGI-скрипту. Применение значения этого свойства внутри JavaScript-программ не оправдано, так как всегда можно получить идентификатор окна или задействовать встроенный массив `frames[0]` и свойства окна `opener`, `top`, `parent` и т.п. Для загрузки внешнего файла в некоторое окно всегда можно применить метод `window.open()`. Но тем не менее, использовать это свойство можно:

```
for(i=1;i<id.frames.length;i++)
{
if(id.frames[i].name==
id.frames[0].document.f0.s0.options[id.frames[0].document.f0.s0.selectedIndex].text)
{
id.frames[i].document.open();
```

```
id.frames[i].document.write("<CENTER>Выбрали этот фрейм</ CENTER>");
id.frames[i].document.close();
}
else
{id.frames[i].document.open();
id.frames[i].document.write("<CENTER>Этот фрейм не выбрали</ CENTER>");
id.frames[i].document.close();
} }
```

В примере организован цикл перебора имен фреймов. Если имя совпадает с указанным именем, то фрейм считается выбранным.

При генерации встроенного в документ объекта Form браузер создает и связанный с ним массив полей формы. Обычно к полям обращаются по имени, но можно обращаться и по индексу массива полей формы:

```
<FORM NAME=fe>
<INPUT NAME=fe1 SIZE=30 MAXLENGTH=30>
<INPUT TYPE=button VALUE="Ввести текст по имени"
   onClick="document.fe.fe1.value='Ввести текст по имени';">
<INPUT TYPE=button VALUE="Ввести текст по индексу"
   onClick="document.fe.elements[0].value='Ввести текст по индексу';">
<INPUT TYPE=reset VALUE="Очистить">
</FORM>
```

Как видно из этого примера, индексирование полей в массиве начинается с цифры "`0`". Общее число полей в форме доступно как результат обращения: document.forms[i].elements.length.

Метод `reset()` (не путать с обработчиком события `onReset`) позволяет установить значения полей формы по умолчанию. При этом использовать кнопку типа Reset не требуется:

```
<FORM NAME=r>
<INPUT VALUE="Значение по умолчанию" SIZE=30 MAXLENGTH=30>
<INPUT TYPE=button VALUE="Изменим текст в поле ввода"
     onClick="document.r.elements[0].value='Изменили текст';">
</FORM>
<A HREF="javascript:document.r.reset();void(0);">
Установили значение по умолчанию</A>
```

В данном примере по гипертекстовой ссылке происходит возврат к форме значения по умолчанию.

Метод `submit()` позволяет проинициировать передачу введенных в форму данных на сервер. При этом методом `submit()` инициируется тот же процесс, что и нажатием на кнопку типа Submit. Это позволяет отложить выполнение передачи данных на сервер:

```
<FORM NAME=s METHOD=post
ACTION="javascript:window.alert('Данные подтверждены');void(0);">
Введите цифру или букву:<INPUT SIZE=1 MAXLENGTH=1>
</FORM>
<A HREF="javascript:document.s.submit();">Отправить данные</A>
```

Вообще говоря, можно написать скрипт, который будет передавать данные без ведома пользователя, с помощью метода `submit()`. Однако браузер выдает предупреждение о таком поведении кода на странице.

Событие reset (восстановление значений по умолчанию в полях формы) возникает при нажатии на кнопку типа Reset или при выполнении метода `reset()`. В теге `FORM` можно переопределить функцию обработки данного события. Для этой цели в него введен атрибут `onReset`:

```
<FORM onReset="javascript:window.alert(
'Event Reset');return false;">
<INPUT VALUE="Значение по умолчанию">
<INPUT TYPE=reset VALUE="Восстановить">
</FORM>
```

В этом примере следует обратить внимание на то, что обработчик события reset возвращает логическое значение `false`. Это сделано для того, чтобы перехватить обработку события reset полностью. Если обработчик события возвращает значение `false`, то установка полей по умолчанию не производится; если обработчик событий возвращает значение `true`, то установка значений полей по умолчанию производится.

Событие submit возникает при нажатии на кнопку типа Submit, графическую кнопку (тип `image`) или при вызове метода `submit()`. Для переопределения метода обработки события submit в тег `FORM` добавлен атрибут *onSubmit*. Функция, определенная в этом атрибуте, будет выполняться перед тем, как отправить данные на сервер. При этом в зависимости от того, что функция вернет в качестве значения, данные либо будут отправлены, либо нет.

```
function test()
{
if(parseInt(document.sub.digit.value).toString()=="NaN")
{
```

```
window.alert("Некорректные данные в поле формы.");
return false;
}
else
{
return true;
}
}
...
<FORM NAME=sub onSubmit="return test();" METHOD=post ACTION="javascript:window.alert('Данные подтверждены');void(0);">
<INPUT NAME=digit SIZE=1 MAXLENGTH=1><INPUT TYPE=submit VALUE="Отправить">
</FORM>
```

В этом примере следует обратить внимание на конструкцию `return test();`. Сама функция `test()` возвращает значения `true` или `false`. Соответственно данные либо отправляются на сервер, либо нет.

Поля ввода (тег `INPUT` типа `TEXT`) являются одним из наиболее популярных объектов программирования на JavaScript. Это объясняется тем, что, помимо использования по прямому назначению, их применяют и в целях отладки программ, вводя в эти поля промежуточные значения переменных и свойств объектов.

```
<FORM>Число гипертекстовых ссылок:
<INPUT SIZE=10 MAXLENGHT=10 VALUE="&{document.links. length};">
до момента обработки формы.
<INPUT TYPE=button VALUE="Число всех гипертекстовых ссылок в документе"
onClick="window.document.forms[0].elements[0].value=document.links. length;">
<INPUT TYPE=reset VALUE="Установить по умолчанию">
</FORM>
```

В данном примере первое поле формы – это поле ввода. Используя подстановку, мы присваиваем ему значение по умолчанию, а потом при помощи кнопки изменяем это значение.

Объект `Text` (текстовое поле ввода) характеризуется следующими свойствами, методами и событиями:

Свойства	Методы	События
DefaultValue	Blur()	OnBlur
Form	Focus()	OnChange
Name	Select()	OnFocus
Type		
Value		

Свойства объекта `Text` – это стандартный набор свойств поля формы. В полях ввода можно изменять только значение свойства `value`.

Обычно при программировании полей ввода решают две типовых задачи: защита поля от ввода данных пользователем и реакция на изменение значения поля ввода.

Для защиты поля от ввода в него символов применяют метод `blur()` в сочетании с обработчиком события `onFocus`:

```
<FORM>
<INPUT SIZE=10 VALUE="1-е значение"
    onFocus="document.forms[0].elements[0].blur();">
<INPUT TYPE=button VALUE=Change
    onClick="document.forms[0].elements[0].value=
'2-е значение';">
<INPUT TYPE=reset VALUE=Reset>
</FORM>
```

В этом примере значение поля ввода можно изменить, только нажав на кнопки Change и Reset. При попытке установить курсор в поле ввода он немедленно оттуда убирается и, таким образом, значение поля не может быть изменено пользователем.

Реакция на изменение значения поля ввода обрабатывается посредством программы, указанной в атрибуте onChange:

```
<FORM METHOD="post" onSubmit="return false;">
<INPUT SIZE="15" MAXLENGHT="15" VALUE="Тест"
    onChange="window.alert(document.forms[0].elements[0].value);">
<INPUT TYPE="button" VALUE="Изменить"
    onClick="document.forms[0].elements[0].value='Change';">
</FORM>
```

Если установить фокус на поле ввода и нажать Enter, ничего не произойдет. Если ввести что-либо в расположенное выше поле ввода, а потом нажать на Enter, то появится окно предупреждения с введенным текстом (для Netscape Navigator) или ничего не произойдет (для Internet Explorer последних версий). Если вы используете Internet Explorer последних версий, то окно предупреждения появится только после установки фокуса вне поля ввода. Во-первых, обработчик onChange вызывается только тогда, когда ввод в поле закончен. Событие не вызывается при каждом нажатии на кнопки клавиатуры при вводе тек-

ста в поле. Во-вторых, обработчик события не вызывается при изменении значения атрибута `value` из JavaScript-программы. В этом можно убедиться, нажав на кнопку Change, – окно предупреждения не открывается. Но если ввести что-то в поле, а после этого нажать на Change, окно появится.

Отметим, что он работает по-разному для Internet Explorer и Netscape Navigator, а именно по-разному обрабатывается событие onChange. Для Internet Explorer при любом изменении поля событие обрабатывается незамедлительно, для Netscape Navigator – после потери фокуса активным полем.

Одним из важных элементов интерфейса пользователя является меню. В HTML-формах для реализации меню используются поля типа `Select` (тег `SELECT`, который, в свою очередь, вмещает в себя теги `Option`). Эти поля представляют собой списки вариантов выбора. При этом список может "выпадать" или прокручиваться внутри окна. Поля типа `Select` позволяют выбрать из списка только один вариант, либо отметить несколько вариантов. Для управления полями типа `Select` в JavaScript существуют объекты `Select` и `Option`.

Эти объекты характеризуются следующими свойствами, методами и событиями:

Объект Select

Свойства	Методы	События
Form Length Name Options[] SelectedIndex Type	Blur() Click() Focus()	OnBlur OnChange OnFocus

Объект Option

Свойства	Методы	События
DefaultSelected Index Selected Text SelectedIndex Value	Нет	Нет

Остановимся на типичных способах применения комбинаций этих двух объектов. Несмотря на то что объект `Option` в нашей таблице находится ниже, что отражает его подчиненное по отношению к `Select` положение, начнем с описания его свойств и особенностей.

Объект `Option` интересен тем, что в отличие от многих других объектов JavaScript имеет конструктор. Это означает, что программист может сам создать объект `Option`:

opt = new Option([text, [value, [defaultSelected, [selected]]]]);

где

`text` – строка текста, которая размещается в теге <LI> (<LI>текст);

`value` – значение, которое передается серверу при выборе альтернативы, связанной с объектом `Option`;

`defaultSelected` – альтернатива выбрана по умолчанию (`true/false`);

`selected` – альтернатива выбрана (`true/false`)

На первый взгляд не очень понятно, для чего может понадобиться программисту такой объект, ведь создать объект типа `Select` нельзя и, следовательно, нельзя приписать ему новый объект `Option`. Все объясняется, когда речь заходит об изменении списка альтернатив встроенных в документ объектов `Select`. Делать это можно, так как изменение списка альтернатив `Select` не приводит к переформатированию документа. Изменение списка альтернатив позволяет решить проблему создания вложенных меню, которых нет в HTML-формах, путем *программирования обычных меню* (`options[]`).

При программировании альтернатив (объект `Option`) следует обратить внимание на то, что среди свойств `Option` нет свойства `name`. Это означает, что к объекту нельзя обратиться по имени. Отсутствие свойства объясняется тем, что у тега `Option` нет атрибута `Name`. К встроенным в документ объектам `Option` можно обращаться только как к элементам массива `options[]` объекта `Select`.

Массив `options[]` – это свойство объекта `Select`. Элементы этого массива обладают теми же свойствами, что и объекты `Option`. Собственно, это и есть объекты `Option`, встроенные в документ. Они создаются по мере загрузки страницы браузером. Программист имеет возможность не только создавать новые объекты `Option`, но и удалять уже созданные браузером объекты:

```
<FORM NAME=f0>
<SELECT NAME=s0>
<OPTION>Первый вариант
<OPTION>Второй вариант
<OPTION>Третий вариант
</SELECT>
<INPUT TYPE=button VALUE="Удалить последний вариант"
```

```
    onClick="document.f0.s0.options[document.f0.s0.length-1]=null;">
<INPUT TYPE=reset VALUE=Reset>
</FORM>
```

В данном примере при загрузке страницы с сервера определены три альтернативы. Они появляются, если выбрать поле `Select`. После нажатия на кнопку удаления последнего варианта (`"Delete last option"`) остаются только две альтернативы. Если еще раз нажать на кнопку удаления альтернативы, останется только одна альтернатива и т.д. В конечном счете, вариантов может не остаться вообще, т.е. пользователь лишится возможности выбора. Кнопка Reset показывает, что альтернативы утеряны бесследно, так как после нажатия на эту кнопку содержание поля `Select` не восстанавливается.

Теперь, используя конструктор `Option`, сделаем процесс обратимым:

```
function def_f1()
{
document.f1.s1.options[0] = new Option("вариант Один","",true,true);
document.f1.s1.options[1] = new Option("вариант Два");
document.f1.s1.options[2] = new Option("вариант Три");
return false;
}
...
<FORM NAME=f1 onReset="def_f1();">
<SELECT NAME=s1>
<OPTION>вариант Один
<OPTION>вариант Два
<OPTION>вариант Три
</SELECT>
<INPUT TYPE=button VALUE="Удалить последний вариант"
    onClick="document.f1.s1.options[document.f1.s1.length-1]=null;">
<INPUT TYPE=reset VALUE=Reset>
</FORM>
```

В данном случае мы обрабатываем событие Reset (тег `Form`). При этом создаем новые объекты типа `Option` и подчиняем их объекту `Select`. При этом первый элемент массива должен быть выбран по умолчанию, чтобы смоделировать поведение при начальной загрузке страницы.

В HTML-формах нельзя реализовать подменю. JavaScript позволяет обойти это ограничение и выполнить замену путем программирования поля `Select`.

В примерах перепрограммирования `options[]` активно используется свойство объекта `Select length`. Оно определяет количество альтернатив, заданных для поля выбора. При помощи этого свойства можно удалять и восстанавливать списки.

Определим посредством этого свойства число вариантов в предыдущем примере:

```
<FORM NAME=f3>
Число вариантов: <INPUT NAME=i0 SIZE=1 MAXLENGTH=1 onFocus="out();">
</FORM>
<SCRIPT>
document.f3.i0.value=document.f1.s1.length;
</SCRIPT>
```

Обратите внимание на тег `SCRIPT`. Он расположен вслед за формой. Если его ввести раньше, то поля формы будут не определены, и в результате мы получим сообщение об ошибке.

Свойство объекта `Select`, которое возвращает значение выбранного варианта, обозначается как `selectedIndex`.

```
<FORM>
Вариант:
<SELECT NAME=s0 onChange = "form.elements[1].value = selectedIndex;">
<OPTION>Один
<OPTION>Два
</SELECT>
Выбрали индекс:
<INPUT SIZE=1 maxlength=1>
</FORM>
```

Посмотрите, как мы обращаемся к элементам текущей формы. Во-первых, мы используем имя Form. Оно указывает на объект Form, к которому принадлежит поле. Во-вторых, мы ссылаемся на второй элемент формы. На данный момент он не определен, но событие произойдет только тогда, когда мы будем выбирать вариант. К этому моменту поле уже будет определено. В-третьих, мы ссылаемся на selectedIndex, не указывая полного имени формы. В данном контексте он относится к текущей форме.

Событие change наступает в тот момент, когда изменяется значение выбранного индекса в объекте `Select`. С изменением этого индекса в полях выбора единственного варианта на данной странице мы сталкивались неоднократно (`selectedIndex` и `options[]`). Данное событие обрабатывается JavaScript-программой, которая указывается в атрибуте onChange тега `Select`. В этом контексте интересно посмотреть, что происходит, когда мы имеем дело с `multiple` тегом `Select`:

```
<FORM>
Набор канцелярских товаров:
<SELECT onChange="form.elements[1].value=";
for(i=0;i<form.elements[0].length;i++)
if(form.elements[0].options[i].selected==true)
form.elements[1].value = form.elements[1].value+i;"multiple>
<OPTION>Вариант 1
<OPTION>Вариант 2
<OPTION>Вариант 3
<OPTION>Вариант 4
<OPTION>Вариант 5
<OPTION>Вариант 6
<OPTION>Вариант 7
</SELECT>
Выбраны позиции:
<INPUT NAME=s1 SIZE=7 MAXLENGTH=7
onFocus="form.elements[1].blur();">
</FORM>
```

Обратите внимание на то, что событие `change` имеет место тогда, когда происходит выбор или отмена альтернативы. Исключение составляет только тот случай, когда варианты при выборе последовательно отмечаются. В этом случае событие происходит в тот момент, когда пользователь отпускает кнопку мыши и все отмеченные альтернативы становятся выбранными.

Свойство `selected` объекта `Option`, на котором был построен пример с канцелярскими принадлежностями, может принимать два значения: `истина (true)` или `ложь (false)`. В примере мы распечатываем индекс выбранной альтернативы, если значение свойства `selected` у объекта `Option` – `true`:

```
if(form.elements[0].options[i].selected==true)
 ...
```

Вообще говоря, свойство `selected` интересно именно в случае поля множественного выбора. В случае выбора единственного варианта его можно получить, указав на свойство `selectedIndex` объекта `Select`.

Свойство `text` представляет собой отображаемый в меню текст, который соответствует альтернативе:

```
<SELECT onChange= "form.elements[2].value=
form.elements[0].options [form.elements[0
].selectedIndex].text;">
</SELECT>
```

В данном примере свойство `text` выводится в текстовое поле формы.

При передаче данных от браузера к серверу в запросе передается текст выбранной опции, если не было указано значение в атрибуте `Value` тега `Option`.

Кнопки. Использование кнопок в Web вообще немыслимо без применения JavaScript.

Кнопка вводится в форму главным образом для того, чтобы можно было обработать событие click:

```
<FORM>
<INPUT TYPE=button VALUE="Окно предупреждения"
    onClick="window.alert('Открыли окно');">
</FORM>
```

Текст, отображаемый на кнопке, определяется атрибутом `Value` тега `INPUT`. С этим атрибутом связано свойство `Value` объекта `Button`. Любопытно, что, согласно спецификации, изменять значение данного атрибута нельзя. Однако в версии 4 Netscape Navigator и Internet Explorer это допустимо.

Следует отметить, что в Netscape Navigator размер кнопки фиксирован (первое значение должно быть самым длинным, иначе будет не очень красиво), а в Internet Explorer размер изменяется в зависимости от длины текста.

Картинки. Кнопки-картинки – это те же кнопки, но только с возможностью отправки данных на сервер. Собственно, такие кнопки в JavaScript составляют две разновидности тега `INPUT`: `image` и `submit`. В JavaScript объект, связанный с данными кнопками, называется Submit.

```
<FORM>
Активная кнопка:
<INPUT TYPE=image SRC=images.gif onClick="return false;">
</FORM>
```

Данный объект обладает теми же свойствами, методами и событиями, что и объект `Button`. Но вот реакция в разных браузерах при обработке событий может быть различной. Так, в событии onClick в Internet Explorer можно отменить передачу данных на сервер, выдав в качестве значения возврата `false`. Netscape Navigator на такое поведение обработчика события вообще не реагирует, и отменять передачу можно только в атрибуте onSubmit тега `Form`:

```
<FORM onSubmit="return false">
```

Активная кнопка:
<INPUT TYPE=image SRC=images.gif border=0>
</FORM>

Наиболее интересной особенностью графических кнопок является их способность передавать в запросе на сервер координаты точки, которую указал пользователь, нажимая на кнопку мышью. К сожалению, обработать такое поведение кнопки в JavaScript-программе не удается.

Передача данных на сервер из формы осуществляется по событию Submit. Это событие происходит при одном из следующих действий пользователя:

- нажата кнопка Submit;
- нажата графическая кнопка;
- нажата клавиша Enter в форме из одного поля;
- вызван метод `submit()`.

Кнопка Submit. Кнопка Submit представляет собой разновидность поля ввода. Она ведет себя так же, как и обычная кнопка, но только еще генерирует событие submit (передачу данных на сервер). В этом, с точки зрения JavaScript-программирования, она абсолютно идентична графическим кнопкам:

<FORM>
<INPUT TYPE=submit VALUE=submit>
</FORM>

В данном примере мы просто перезагружаем страницу.

С точки зрения программирования наибольший интерес представляет возможность перехвата события submit и выполнение при этом действий, отличных от стандартных. Для этой цели у кнопки есть атрибут обработки события click (onClick):

<FORM>
<INPUT TYPE=submit VALUE=Submit onClick="return false;">
</FORM>

При нажатии на кнопку перезагрузки страницы не происходит – передача данных на сервер отменена. Обработчик действует так же, как обработчик события submit в теге `Form`.

Теперь можно написать собственную программу обработки события submit:

```
function my_submit()
{
if(window.confirm("Хотите перегрузить страницу?")) return true;
else return false;
}
...
<FORM>
<INPUT TYPE=submit VALUE=Submit onClick="return my_submit();">
</FORM>
```

Если подтвердить необходимость перезагрузки страницы, она действительно будет перезагружена, а при отказе (cancel) вы вернетесь в текущую страницу без перезагрузки. Действия могут быть и более сложными. В любом случае, если функция обработки возвращает значение `true`, то передача данных на сервер (в нашем примере – перезагрузка страницы) происходит, иначе (значение `false`) – данные не передаются.

Единственное поле в форме. Если в форме присутствует одно-единственное поле, и мы в него осуществили ввод и после этого нажали Enter, то браузер сгенерирует событие submit:

<FORM onSubmit="window.alert('Сделано');return false;">
<INPUT SIZE=10 MAXLENGTH=10>
</FORM>

Перехватить такое событие и обработать можно только за счет программы обработки события submit в теге `Form`, что и сделано в примере.

В этом примере, кроме поля ввода, в форме присутствует меню. Если менять значения выбранных альтернатив, то перезагрузки не происходит, но стоит изменить значение в поле ввода и нажать Enter, происходит submit, и система выдает окно предупреждения.

Метод submit()

Метод `submit()` – это метод формы. Если в программе вызывается метод `submit`, то данные из формы, к которой применяется данный метод, передаются на сервер. Усовершенствуем пример с полем ввода и меню выбора (прежде чем выбирать альтернативы, прочтите комментарий под примером):

<FORM onSubmit="window.alert('Сделано');return false;">
<INPUT SIZE=10 MAXLENGTH=10>
<SELECT onChange="form.submit();">
<OPTION>Вариант 1<OPTION>Вариант 2</SELECT>
</FORM>

При выборе альтернативы пользователь сразу инициирует обмен данными с сервером. Событие submit в данном случае обработчиком событий не перехватывается, в отличие от нажатия Enter. Такое поведение браузера довольно логично. Если программист вызвал метод `submit()`, то, наверное, он предварительно проверил данные, которые отправляет на сервер.

Cookies. Cookies не являются полями формы, но, тем не менее, отойдя от строгого рассмотрения иерархии объектов JavaScript, мы уделим им немного внимания как одному из механизмов управления обменом данных. Основная функция `cookie` – поддержка сеанса работы между клиентом (браузером) и сервером.

Cookie – это небольшой фрагмент текста, который передается от сервера браузеру и потом может быть возвращен обратно. Программа на JavaScript способна прочитать выставленное значение cookie и даже изменить его. Для этой цели используют свойство объекта `Document` – `cookie`:

```
<FORM>
<INPUT TYPE=button VALUE="Показать Cookies"
onClick="window.alert(window.document.cookie);">
</FORM>
```

В данном случае cookies отображаются в виде одной большой строки со множеством значений. Свойство `cookie` документа можно переопределить:

```
function asign()
{
document.cookie="n1=3";
window.alert(document.cookie);
}
...
<FORM>
<INPUT TYPE=button VALUE="Изменить n1" onClick="asign()">
</FORM>
```

Как видно из примера, программисту не нужно выделять `cookie` из строки. Браузер рассматривает cookies как ассоциированный массив (хеш) и изменяет значение `cookie` по его имени.

Наконец, `cookie` можно удалить. Если быть более точным – деактивировать, указав время его действия:

```
function change_cookies()
{
a = new Array();
c = new Date();
a = document.cookie.split(';');
document.cookie=a[0]+"; expires="+c.toGMTString()+";";
window.alert(document.cookie);
}
...
<FORM>
<INPUT TYPE=button VALUE="delete cookies" onClick="change_cookies()">
</FORM>
```

В данном случае мы "удаляем" `cookie` за счет параметра `expire` (времени, до которого `cookie` живет). Так как мы берем текущее время, то `cookie` исчезает из списка. Многократно нажимая на кнопку, можно удалить все cookies для данной страницы.

4.7. ПРОГРАММИРОВАНИЕ ГРАФИКИ

Наиболее зрелищные эффекты при программировании на JavaScript достигаются при работе с графикой. Программирование графики в JavaScript опирается на объект `Image`, который характеризуется следующими свойствами, методами и событиями:

Свойства	Методы	События
Border Complete Height Hspace Name Src Vspace Width Lowsrc	Нет	OnAbort OnError OnLoad

Несмотря на такое обилие свойств, их абсолютное большинство можно только читать, но не изменять. Об этом свидетельствует, прежде всего, отсутствие методов. Но два свойства все же можно изменять: `src` и `lowsrc`. Этого оказывается достаточно для множества эффектов с картинками.

Все объекты класса `Image` можно разделить на встроенные и поро-жденные программистом. Встроенные объекты – это картинки тегов IMG. Если эти картинки поименовать, к ним можно обращаться по имени:

```
<A HREF="javascript:void(0);"
onClick="window.alert('Image name:'+
      document.images[0].name)">
<IMG NAME=tstu SRC=images.gif BORDER=0>
</A>
```

Картинка активна. Если на нее нажать, получим имя тега IMG. Обращение `document.images[0].name` позволяет распечатать это имя в окне предупреждения. При этом само имя указано как `name=tstu` в теге IMG.

К встроенному графическому объекту можно обратиться и по индексу:

```
document.images[0];
```

В данном случае images[0] – это первая картинка документа.

Свойства `src` и `lowsrc` определяют URL изображения, которое монтируется внутрь документа. При этом `lowsrc` определяет временное изображение, обычно маленькое, которое отображается, пока загружается основное изображение, чей URL указывается в атрибуте `SRC` тега IMG. Свойство `src` принимает значение атрибута `SRC` тега IMG. Программист может изменять значения и `src`, и `lowsrc`. Рассмотрим пример с `src`:

```
document.i2.src="images2.gif";
```

Как видно из этого примера, существует возможность модифицировать вмонтированную картинку за счет изменения значения свойства src встроенного объекта Image. Если вы в первый раз просматриваете данную страницу (т.е. картинки не закешированы браузером), то постепенное изменение картинки будет заметно.

Изменение картинки. Изменить картинку можно, только присвоив свойству `src` встроенного объекта `Image` новое значение. Очевидно, что медленная перезагрузка картинки с сервера не позволяет реализовать быстрое листание. Попробуем решить эту проблему.

Собственно, решение заключается в разведении по времени подкачки картинки и ее отображения. Для этой цели используют конструктор объекта `Image`:

```
<TABLE>
<TD>
<A HREF="javascript:void(0)";
onMouseover="document.m0.src=color[0].src;
          return true;"
onMouseout="document.m0.src=mono[0].src;
         return true;">
<IMG NAME=m0 SRC=images0.gif border=0>
</A>
</TD>
...
</TABLE>
```

Фрагмент кода показывает типовой прием замещения и восстановления картинки при проходе курсора мыши. Естественно, что менять можно не одну, а сразу несколько картинок.

Главное, тем не менее, не в том, что картинки замещаются, а в том, с какой скоростью они это делают. Для достижения нужного результата в начале страницы создаются массивы картинок, в которые перед отображением перекачивается графика:

```
color = new Array(32);
mono = new Array(32);
for(i=0;i<32;i++)
{ mono[i] = new Image();
color[i] = new Image();
if(i.toString().length==2)
{
mono[i].src = "images0"+i+".gif";
color[i].src = "images0"+i+".gif";
}
else
{
mono[i].src = "images0"+i+".gif";
color[i].src = "images0"+i+".gif";
}
}
```

Еще один характерный прием – применение функции отложенного исполнения JavaScript-кода (`eval()`):

```
function def()
{
for(i=0;i<32;i++)
{
eval("document.m"+i+".src=mono["+i+"].src");
}
for(i=0;i<5;i++)
{
eval("document.r"+i+".src=rm["+i+"].src");
}
}
```

В данном случае `eval()` избавляет нас от необходимости набирать операции присваивания.

Мультипликация. Естественным продолжением идеи замещения значения атрибута `SRC` в теге IMG является мультипликация, т.е. последовательное изменение значения этого атрибута во времени. Для реализации мультипликации используют метод объекта `Window` – `setTimeout()`.

Собственно, существуют два способа запуска мультипликации:

- onLoad();
- onClick(), onChange() ...

Наиболее популярный – `setTimeout()` при `onLoad()`.

Событие `onLoad()` наступает в момент окончания загрузки документа браузером. Обработчик события указывается в теге `BODY`:

```
...
<BODY onLoad="JavaScript_code">
...
```

В нашем случае при загрузке документа должен начать выполняться цикл изменения картинки:

```
function movie()
{
eval("document.images[0].src='clock"+
    i+".gif';");
i++;if(i>6) i=0;
setTimeout("movie();",500);
}
...
<BODY onLoad="movie();">
...
```

В примере используется бесконечный цикл, хотя можно реализовать и конечное число подмен:

```
function movie()
{
eval("document.images[0].src='clock"+
    i+".gif';");
i++;
if(i<7)
{
setTimeout("movie();",500);}
}
...
<BODY onLoad="movie();">
```

В обоих примерах следует обратить внимание на использование метода `setTimeout()`. На первый взгляд, это просто рекурсия. Но в действительности все несколько сложнее. JavaScript разрабатывался для многопоточных операционных систем, поэтому правильнее будет представлять себе исполнение скриптов следующим образом:

1. Скрипт получает управление при событии `onLoad()`.
2. Заменяет картинку.
3. Порождает новый скрипт и откладывает его исполнение на 500 миллисекунд.
4. Текущий скрипт уничтожается JavaScript-интерпретатором.

После окончания срока задержки исполнения все повторяется. В первом примере (бесконечное повторение) функция порождает саму себя и тем самым поддерживает непрерывность своего выполнения. Во втором примере (конечное число итераций) после девяти повторов функция не порождается. Это приводит к завершению процесса отображения новых картинок.

Перманентная мультипликация может быть достигнута и другими средствами, например многокадровыми графическими файлами. Однако движение на странице – не всегда благо. Часто возникает желание реализовать запуск и останов движения по требованию пользователя. Удовлетворим это желание, используя предыдущие примеры (запустить или остановить мультипликацию):

```
var flag1=0;
function movie()
{
if(flag1==0)
{
eval("document.images[0].src='clock"+
     i+".gif';");
i++;if(i>6) i=0;
}
setTimeout("movie();",500);
}
...
<BODY onLoad="movie();">
...
<FORM>
<INPUT TYPE=button VALUE="Start/Stop"
onClick="if(flag1==0) flag1=1; else flag1=0;">
</FORM>
```

В данном случае мы просто обходим изменение картинки, но при этом не прекращаем порождение потока. Если мы поместим `setTimeout()` внутрь конструкции `if()`, то после нажатия на кнопку Start/Stop поток порождаться не будет, и запустить движение будет нельзя.

Существует еще один способ решения проблемы остановки и старта мультипликации. Он основан на применении метода `clearTimeout()`. Внешне все выглядит по-прежнему, но процесс идет совсем по-другому:

```
var flag1=0;
var id1;
function movie()
{
eval("document.images[0].src='clock"+
     i+".gif';");
i++;if(i>6) i=0;
id1 = setTimeout("movie();",500);
}
...
<BODY onLoad="movie();">
...
<FORM>
<INPUT TYPE=button VALUE="Start/Stop"
onClick="if(flag1==0)
  { id1=setTimeout('movie();',500); flag1=1;}
  else {clearTimeout(id1); flag1=0;};">
</FORM>
```

Обратите внимание на два изменения. Во-первых, объявлен и используется идентификатор потока (`id1`); во-вторых, применяется метод `clearTimeout()`, которому, собственно, идентификатор потока и передается в качестве аргумента. Чтобы остановить воспроизведение функции `movie()`, достаточно "убить" поток.

При программировании графики следует учитывать множество факторов, которые влияют на скорость отображения страницы и скорость изменения графических образов. При этом обычная дилемма оптимизации программ – скорость или размер занимаемой памяти – решается только путем увеличения скорости. О размере памяти при программировании на JavaScript думать как-то не принято.

Из всех способов оптимизации отображения картинок мы остановимся только на нескольких:

- оптимизация отображения при загрузке;
- оптимизация отображения за счет предварительной загрузки;
- оптимизация отображения за счет нарезки изображения.

Если первые две позиции относятся в равной степени как к отображению статических картинок, так и к мультипликации, то третий пункт характерен главным образом для мультипликации.

Практически в любом руководстве по разработке HTML-страниц отмечается, что при использовании тега IMG в теле HTML-страницы следует указывать атрибуты `WIDTH` и `HEIGHT`. Это продиктовано порядком загрузки компонентов страницы с сервера и алгоритмом работы HTML-парсера. Первым загружается текст разметки. После этого парсер разбирает текст

и начинает загрузку дополнительных компонентов, в том числе графики. При этом загрузка картинок, в зависимости от типа HTTP-протокола, может идти последовательно или параллельно.

Также параллельно с загрузкой парсер продолжает свою работу. Если для картинок заданы параметры ширины и высоты, то можно отформатировать текст и отобразить его в окне браузера. До тех пор, пока эти параметры не определены, отображения текста не происходит.

Таким образом, указание высоты и ширины картинки позволит отобразить документ раньше, чем картинки будут получены с сервера. Это дает пользователю возможность читать документ или задействовать его гипертекстовые ссылки до момента полной загрузки (событие `load`).

С точки зрения JavaScript, указание размеров картинки задает начальные параметры окна отображения графики внутри документа. Это позволяет воспользоваться маленьким прозрачным образом для того, чтобы заменить его полноценной картинкой. Идея состоит в передаче маленького объекта для замещения его по требованию большим объектом.

Замена одного образа другим часто бывает оправдана только в том случае, когда это происходит достаточно быстро. Если перезагрузка длится долго, то эффект теряется. Для быстрой подмены используют возможность предварительной загрузки документа в специально созданный объект класса `Image`.

Реальный эффект можно почувствовать только при отключении кэширования страниц на стороне клиента (браузера). Кэширование часто используют для ускорения работы со страницами Web-узла. Как правило, загрузка первой страницы – это достаточно длительный процесс. Самое главное, чтобы пользователь в этот момент был готов немного подождать. Поэтому, кроме графики, необходимой только на первой странице, ему можно передать и графику, которая на ней не отображается. Но зато при переходе к другим страницам узла она будет отображаться без задержки на передачу с сервера.

Нарезка картинок применяется довольно часто. Она позволяет достигать эффекта частичного изменения отображаемой картинки. Чаще всего он применяется при создании меню.

Кроме подобного эффекта нарезка позволяет реализовать мультипликацию на больших картинках. При этом изменяется не весь образ, а только отдельные его части.

Одним из наиболее популярных приемов дизайна страниц Web-узла является техника нарезки картинок на составные части. Можно выделить следующие способы применения этой техники для организации навигационных компонентов страницы:

- горизонтальные и вертикальные меню;
- вложенные меню;
- навигационные графические блоки.

Главной проблемой при использовании нарезанной графики является защита ее от контекстного форматирования страницы HTML-парсером. Дело в том, что он автоматически переносит элементы разметки на новую строку, если они не помещаются в одной. Составные части нарезанной картинки должны быть расположены определенным образом, поэтому простое их перечисление в ряд не дает желаемого эффекта (рис. 70):

```
<IMG SRC="image1.gif"><IMG
SRC="image2.gif"><IMG
SRC="image3.gif"><IMG
SRC="image4.gif">
```

Рис. 70

Элементы переносятся на новую строку, так как ширина раздела меньше общей ширины всех картинок. Проблема решается, если применить защиту от парсера – `<PRE>` (рис. 71):

```
<PRE>
<IMG SRC="image1.gif"><IMG
SRC="image2.gif"><IMG
SRC="image3.gif"><IMG
SRC="image4.gif">
</PRE>
```

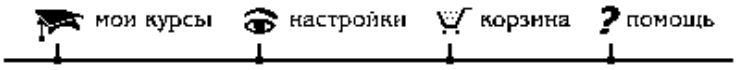

Рис. 71

Использование такого меню требует определения на нем гипертекстовых ссылок, что приводит к следующему эффекту (рис.72):

```
<PRE>
<A HREF="javascript:void(0);"><IMG
SRC="image1.gif"></A><A
```

```
HREF="javascript:void(0);"><IMG
SRC="image2.gif"></A><A
HREF="javascript:void(0);"><IMG
SRC="image3.gif"></A><A
HREF="javascript:void(0);"><IMG
SRC="image4.gif"></A>
</PRE>
```

Рис. 72

Этого можно достичь за счет применения атрибута `BORDER`, равного 0 (рис. 73):

```
<PRE>
<A HREF="javascript:void(0);"><IMG
SRC="image1.gif" BORDER="0"></A><A
HREF="javascript:void(0);"><IMG
SRC="image2.gif" BORDER="0"></A><A
HREF="javascript:void(0);"><IMG
SRC="image3.gif" BORDER="0"></A><A
HREF="javascript:void(0);"><IMG
SRC="image4.gif" BORDER="0"></A>
</PRE>
```

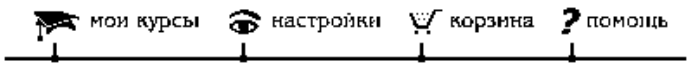

Рис. 73

Теперь попробуем тем же способом реализовать многострочное меню (рис. 74):

```
<PRE>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image1.gif" WIDTH=103 HEIGHT=21 BORDER=0></A>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image2.gif" WIDTH=103 HEIGHT=21 BORDER=0></A>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image3.gif" WIDTH=103 HEIGHT=21 BORDER=0></A>
<IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0><A
HREF="javascript:void(0);"><IMG SRC="image4.gif" WIDTH=103 HEIGHT=21 BORDER=0></A>
</PRE>
```

Рис. 74

Сплошной картинки не получается, так как высота строки не равна высоте картинки. Подогнать эти параметры практически невозможно. Каждый пользователь настраивает браузер по своему вкусу. Решение заключается в использовании таблицы (рис. 75):

```
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 ALIGN=center>
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image1.gif" WIDTH=103 HEIGHT=21 BORDER=0></A></TD>
</TR>
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image2.gif" WIDTH=103 HEIGHT=21 BORDER=0></A></TD>
</TR>
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image3.gif" WIDTH=103 HEIGHT=21 BORDER=0></A></TD>
</TR>
```

```
<TR>
<TD><IMG SRC=tree.gif WIDTH=27 HEIGHT=21 BORDER=0></TD>
<TD><A HREF="javascript:void(0);"><IMG SRC="image14.gif" WIDTH=103 HEIGHT=21 BORDER=0></A></TD>
</TR>
</TABLE>
```

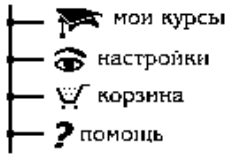

Рис. 75

В данном случае все картинки удается сшить без пропусков и тем самым достичь непрерывности навигационного дерева. Пропуски устраняются путем применения атрибутов `border`, cellspacing и cellpadding.

Остановимся на наиболее типичном способе комбинирования обработчиков событий и изменения графических образов. Продолжая обсуждение примера с навигационным деревом, покажем его развитие с обработкой событий, вызванных наведением мыши на объект, и изменением картинок:

```
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 ALIGN=center>
<TR>
<TD><IMG SRC=image.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
<TD><IMG SRC=image1.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
</TR>
<TR>
<TD><IMG SRC=image2.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.manual.src='image3.gif';return true;"
onMouseout="document.manual.src='image4.gif'; return true;">
<IMG SRC=image5.gif BORDER=0 WIDTH=20 HEIGHT=20></A></TD>
</TR>
<TR>
<TD><IMG SRC=image6.gif WIDTH=20 HEIGHT=20 BORDER=0></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.desk.src='image7.gif';return true;"
onMouseout="document.desk.src='image8.gif';return true;">
<IMG SRC=image9.gif BORDER=0 WIDTH=20 HEIGHT=20></A></TD>
</TR>
</TABLE>
```

В данном примере при проходе курсор мышки через картинки меню последние изменяются. Этот эффект достигается за счет применения двух событий: `onMouseover` и `onMouseout`. По первому событию картинка меняется с позитива на негатив, по второму событию восстанавливается первоначальный вариант. Следует заметить, что события определены в теге якоря (`A`), а не в теге IMG. Это наиболее устойчивый с точки зрения совместимости браузеров вариант.

Вертикальные и горизонтальные меню. Графическое меню удобно тем, что автор может всегда достаточно точно расположить его компоненты на экране. Это, в свою очередь, позволяет и другие элементы страницы точнее располагать относительно элементов меню (рис. 76):

```
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">
<TR ALIGN="center">
<TD><IMG NAME="e0" SRC="empty.gif" WIDTH="15" HEIGHT="8" BORDER="0"></TD>
<TD><IMG NAME="e1" SRC="empty.gif" WIDTH="15" HEIGHT="8" BORDER="0"></TD>
<TD><IMG NAME="e2" SRC="empty.gif" WIDTH="15" HEIGHT="8" BORDER="0"></TD>
<TD><IMG NAME="e3" SRC="empty.gif" WIDTH="15" HEIGHT="8" BORDER="0"></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="document.e0.src= 'arrowdw.gif';return true;"
 onMouseout="document.e0.src='empty.gif';return true;">
 <IMG SRC="image1.gif" BORDER="0"></A></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.e1.src= 'arrowdw.gif';return true;"
onMouseout="document.e1.src='empty.gif';return true;">
<IMG SRC="image2.gif" BORDER="0"></A></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.e2.src= 'arrowdw.gif';return true;"
onMouseout="document.e2.src='empty.gif';return true;">
<IMG SRC="image3.gif" BORDER="0"></A></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.e3.src= 'arrowdw.gif';return true;"
 onMouseout="document.e3.src='empty.gif';return true;">
 <IMG SRC="image4.gif" BORDER="0"></A></TD>
</TR>
```

</TABLE>

Рис. 76

В данном случае стрелочка бежит точно над тем элементом, на который указывает мышь. По большому счету, применение атрибута ALT у IMG и его дублирование в строке статуса является гораздо более информативным, чем добавление нового графического элемента. Правда, отображается содержание ALT с некоторой задержкой (рис. 77):

Рис. 77

Посмотрим теперь на реализацию вертикального меню, построенного на основе графических блоков текста (рис. 78):

```
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="document.evente1. src='corner.gif';"
onMouseout="document.evente1.src='clear.gif';">
<IMG SRC="image1.gif" border="0"></A></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.evente1. src='corner.gif';"
onMouseout="document.evente1.src='clear.gif';">
<IMG NAME="evente1" SRC="clear.gif" border="0"></A></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="document.evente2. src='corner.gif';"
onMouseout="document.evente2.src='clear.gif';">
<IMG SRC="image2.gif" border="0"></A></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.evente2. src='corner.gif';"
onMouseout="document.evente2.src='clear.gif';">
<IMG NAME="evente2" SRC="clear.gif" border="0"></A></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="document.evente3. src='corner.gif';"
onMouseout="document.evente3.src='clear.gif';">
<IMG SRC="image3.gif" border="0"></A></TD>
<TD><A HREF="javascript:void(0);" onMouseover="document.evente3. src='corner.gif';"
onMouseout="document.evente3.src='clear.gif';">
<IMG NAME="evente3" SRC="clear.gif" border="0"></A></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="document.evente4. src='corner.gif';"
 onMouseout="document.evente4.src='clear.gif';">
 <IMG SRC="image4.gif" border="0"></A></TD>
<TD>
<A HREF="javascript:void(0);" onMouseover="document.evente4. src='corner.gif';"
onMouseout="document.evente4.src='clear.gif';">
<IMG NAME="evente4" SRC="clear.gif" border="0">
</A></TD>
</TR>
</TABLE>
```

Рис. 78

При движении мыши у соответствующего компонента, попавшего в фокус мыши, "отгибается уголок". В данном случае "уголок" – это самостоятельная картинка. Все уголки реализованы в правой колонке таблицы. Для того чтобы гипертексто-

вая ссылка срабатывала по обеим картинкам (тексту и "уголку"), применяются одинаковые теги A, охватывающие графические образы. В этом решении есть один недостаток: при переходе от текста к "уголку" последний "подмигивает". Картинки можно разместить и в одной ячейке таблицы, но тогда нужно задать ее ширину, иначе при изменении размеров окна браузера картинки могут "съехать". Чтобы убрать "подмигивание", необходимо сделать полноценные картинки замены.

"Подмигивание" происходит при переходе с одного элемента разметки тега на другой. При этом заново определяются свойства отображения элемента.

Вложенные меню. В HTML нет стандартного способа реализации вложенных меню. Тем не менее за счет графики можно создать их подобие. При этом следует понимать, что место, на которое ложится графика, нельзя заполнить текстом (рис. 79):

```
<SCRIPT>
function submenu(a)
{
if(a==1)
{
document.menu00.src="image1.gif"; // 1 (активна)
document.menu10.src="image2.gif"; // 2
document.menu01.src="image3.gif"; // 1 пункт вложенного меню 1
document.menu02.src="image4.gif"; // 2 пункт вложенного меню 1
}
if(a==2)
{
document.menu00.src="image1.gif"; // 2
document.menu10.src="image2.gif"; // 1 (активна)
document.menu01.src="image3.gif"; // 1 пункт вложенного меню 2
document.menu02.src="image4.gif"; // 2 пункт вложенного меню 2
}
}
</SCRIPT>

<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="submenu(1);return true;">
<IMG NAME=menu00 SRC=image1.gif BORDER=0></a></td>
<TD><IMG NAME=menu01 SRC=image3.gif BORDER=0></TD>
</TR>
<TR>
<TD><A HREF="javascript:void(0);" onMouseover="submenu(2);return true;">
<IMG NAME=menu10 SRC=image2.gif BORDER=0></td>
<TD><IMG NAME=menu02 SRC=image4.gif BORDER=0></TD>
</TR>
</TABLE>
```

Рис. 79

В этом примере вложенное меню расположено справа от основного. Эффект вложенности достигается за счет изменения цвета. Подчиненность меню можно подчеркнуть изменением его положения относительно основного меню (рис. 80).

Рис. 80

В этом случае для продвижения меню вниз необходимо зарезервировать место при помощи невидимых или видимых картинок. Это не обязательно должны быть иллюстративные картинки, которые не несут никакой нагрузки.

При использовании слоев можно создать настоящее выпадающее меню.

4.8. ПРОГРАММИРОВАНИЕ ГИПЕРССЫЛОК

Кроме графики, на странице есть еще несколько встроенных массивов объектов, элементы которых можно изменять. Один из них – массив гипертекстовых ссылок.

Гипертекстовая ссылка относится к классу объектов URL. К этому классу объектов относятся:

- `Location`;
- Area;
- Link.

Область применения URL на HTML-страницах гораздо шире этих трех позиций. Чаще всего мы сталкиваемся с ним при программировании:

- графики (атрибут `SRC` контейнера `IMG`);
- форм (атрибут `ACTION` контейнера `Form`);
- ссылок (атрибут HREF контейнера `A`);
- "чувствительных" картинок (атрибут HREF контейнера Area).

В данном разделе рассмотрим программирование собственно гипертекстовых переходов и "чувствительных" картинок.

4.8.1. Объект URL

Объект класса URL обладает свойствами, которые определены схемой URL. В качестве примера рассмотрим ссылку на применение атрибута `SRC` в контейнере `IMG`:

http://tstu.ru/help/index.html

Значения свойств

Href:	http://tstu.ru/help/index.html
Protocol:	http:
Hostname:	intuit.ru
Host:	intuit.ru:80
Port:	80
Pathname:	help/index.html
Search:	
Hash:	

Обращение к свойству объекта класса URL выглядит как:

имя_объект_класса_URL.свойство

Например, так:

document.links[0].href

document.location.host

document.links[2].hash

Свойства объекта URL дают программисту возможность менять только часть URL-объекта (гипертекстовой ссылки, например). Наиболее интересно это выглядит в объекте `Location`, когда при изменении свойства происходит перезагрузка документа. Однако и при работе с обычными гипертекстовыми ссылками такая технология более предпочтительна, чем изменение всего URL целиком.

Здесь следует заметить, что чаще всего все-таки меняют весь URL. Это связано с тем, что такое действие более понятно с точки зрения HTML-разметки. Ведь у контейнера `A` нет атрибута PROTOCOL, но зато есть атрибут HREF.

4.8.2. Массивы встроенных гипертекстовых ссылок

К встроенным гипертекстовым ссылкам относятся собственно ссылки (`<A HREF=...>...</A>`) и ссылки "чувствительных" графических картинок. Они составляют встроенный массив гипертекстовых ссылок документа (`document.links[]`).

К сожалению, обратиться по имени к гипертекстовой ссылке нельзя. Точнее такое обращение не рекомендуется в силу различий между браузерами. Поэтому обращаться к ним можно только как к массиву встроенных ссылок. В качестве примера распечатаем гипертекстовые ссылки некоторого документа:

```
for(i=0;i<document.links.length;i++)
document.write(document.links[i].href+"<BR>");
```

Список ссылок:

http://www.tstu.ru/help/index.html

http://www.tstu.ru/help/terms.html

http://www.tstu.ru/help/shop.html

Вставим в документ контейнер `MAP`:

```
<MAP NAME=test>
<AREA SHAPE=rect COORDS="0,0,0,0"
HREF="javascript:window.alert('Area_Link_1');void(0);">
<AREA SHAPE=rect COORDS="0,0,0,0"
HREF="javascript:window.alert('Area_Link_2');void(0);">
</MAP>
```

И снова распечатаем массив ссылок:
links[0]:http://www.tstu.ru/help/index.html
links[1]:http://www.tstu.ru/help/terms.html
links[2]:http://www.tstu.ru/help/shop.html
links[3]:javascript:window.alert('Area_Link_1');void(0);
links[4]:javascript:window.alert('Area_Link_2');void(0);

Две новые ссылки – это ссылки из контейнера `MAP`, который не отображается, но ссылки из него попадают в массив встроенных ссылок. При этом, как в нашем случае, они могут попасть между обычными гипертекстовыми ссылками, если контейнер `MAP` расположить внутри текста документа. На данной странице он помещен перед контейнером `Script`, в котором мы распечатываем массив встроенных ссылок.

4.8.3. Замена атрибута HREF

Рассмотрим, как при помощи JavaScript-кода можно управлять свойствами объекта класса Link на примере меню типа "записная книжка" (рис. 81):

Рис. 81

Конечно, это не настоящая "записная книжка". Поле формы заполняется только при выборе гипертекстовой ссылки, расположенной над этим полем. Единственная цель данного примера – показать, как изменяется значение атрибута HREF (оно отображается в поле status окна браузера). Изменение производится посредством вызова функции:

```
function line(a)
{
if(a==0)
{
clear();
window.document.o0.src="fio.gif";
window.document.all['lo0'].href='javascript:window.document.f1.fi1.value= "Фамилия И.О."; void(0);';
window.document.o1.src="rpho.gif";
window.document.all['lo1'].href='javascript:window.document.f1.fi2.value= "253-93-10"; void(0);';
window.document.o2.src="hpho.gif";
window.document.all['lo2'].href='javascript:window.document.f1.fi3.value= "253-93-12"; void(0);';
}
.....
}
```

В данном случае мы работаем с тремя элементами массива встроенных гипертекстовых ссылок: `all['lo0']`, `all['lo1']` и `all['lo2']`. У каждого из них при вызове функции со значением аргумента `a`, равным 0, 1 и 2, соответственно, изменяем значение свойства href. Это свойство мы меняем целиком. URL можно менять и частично.

4.8.4. Изменение части URL

Гипертекстовая ссылка – это объект класса URL. У этого объекта можно изменять и другие свойства. Проиллюстрируем эту возможность при частичном изменении ссылки. Распечатаем сначала свойство, которое не зависит от протокола (в нашем случае от `JavaScript`) `document.all.next.pathname: href:--> http://tstu.ru/help/index.html`

pathname:--> help/index.html
Изменим теперь pathname:

```
document.all.next.pathname="test";
document.write(window.document.all.next.pathname);
```

href:--> http://tstu.ru:80/test
pathname--> test

Обратите внимание, что Internet Explorer самостоятельно добавил в ссылку номер порта. По этой причине использовать свойства, отличные от href, в ссылках, где используется схема `JavaScript`, не рекомендуется.

4.8.5. Обработка событий Mouseover и Mouseout

Эти два события изо всех событий, которые обрабатываются на страницах Web, используются чаще всего. Именно они позволяют обесцвечивать и проявлять картинки, а также менять содержание поля status. Первое событие генерируется браузером, если курсор мыши указывает на гипертекстовую ссылку, а второе – когда он покидает гипертекстовую ссылку. Рассмотрим пример с записной книжкой, но только для проявления меню второго уровня будем использовать обработчик события `onMouseover`:

```
<TABLE BORDER=0 CELLSPACING=1 CELLPADDING=0
    ALIGN=center>
<TR>
```

```
<TD>
<A HREF="javascript:void(0);" onMouseover="line2(0); return true;">
<IMG SRC=addrpho.gif BORDER=0></A></TD>
...
</TR>
</TABLE>
```

В качестве обработчика события мы вызываем функцию `line2()`, которая идентична `line1()` из предыдущего примера. В примере `IMG` перенесен на новую строку для наглядности. На самом деле так поступать не следует – при интерпретации HTML-парсером могут появиться неучтенные пропуски, которые не предусмотрены автором страницы (рис. 82).

Рис. 82

Мы рассмотрели редкий пример, в котором не требуется возврата предыдущего значения после прохода мыши по гипертекстовой ссылке. По этой причине в гипертекстовой ссылке не применялся второй обработчик `onMouseout`. В большинстве случаев, например при расцвечивании картинки, он требуется:

```
<A HREF="javascript:void(0);"
onMouseover="document.pic1.src='image2.gif';
        return true;"
onMouseout="document.pic1.src='image.gif';
        return true;">
<IMG NAME=pic1 src=image.gif BORDER=0></A>
```

Рассматривая предыдущий пример, мы не обсудили использование функции return. При работе с графикой значение, которое возвращает обработчик события, на результат отображения не влияет. Но если изменять значение поля статуса браузера, то изменения произойдут только в случае возврата значения `true`.

4.8.6. Схема URL-"javascript:..."

Для программирования гипертекстовых переходов в спецификацию универсального идентификатора ресурсов (URL) разработчики JavaScript ввели отдельную схему по аналогии со схемами `http`, `ftp` и т.п. – `javascript`. Рассмотрим общий случай обработки события гипертекстового перехода при выборе гипертекстовой ссылки.

Схема URL-`javascript` в общем виде выглядит следующим образом:

```
<A HREF="javascript:...;">...</A>
<FORM ACTION="javascript:...">
```

Одним словом, в любом месте, где мы используем URL, вместо любой из стандартных схем можно применить схему `javascript`. Единственное исключение составляет контейнер `IMG`. URL в нем используется в атрибуте `SRC`. Принять определенное значение `SRC` может при помощи либо назначения в `IMG`, либо обращения к свойству `IMG`. По большому счету, применение JavaScript в `SRC` может только проинициализировать картинку. Дальнейшее ее изменение описано в разделе "Программируем графику". Рассмотрим пример простой гипертекстовой ссылки:

```
<A HREF="javascript:window.alert('window.alert() изменяет HREF'); void(0);"> Заменили обычную ссылку
</A>
```

Можно выполнить аналогичную операцию, но над картинкой:

```
var flag=0;
function ichange()
{
if(flag==0)
{
document.i1.src="image1.gif"; flag=1;
}
else
{
document.i1.src="image2.gif"; flag=0;
}
```

```
}
...
<A HREF="javascript:ichange();void(0);">
<IMG NAME=i1 SRC=image2.gif BORDER=0>
</A>
```

Попробуем теперь выполнить JavaScript-код применительно к контейнеру `FORM`:

```
<FORM NAME=f action="javascript:window.alert(
   document.f.fi0.value);void(0);"
METHOD=post>Введите текст для отображения в окне и нажмите ввод:
<INPUT NAME=fi0 SIZE=20 MAXLENGTH=20>
</FORM>
```

Следует отметить, что все-таки использование схемы `javascript` в этом месте HTML-разметки выглядит неудачно. Гораздо логичнее применить обработчик события `onSubmit`.

4.9. ТИПЫ И СТРУКТУРЫ ДАННЫХ

Как и любой другой язык программирования, JavaScript поддерживает встроенные типы и структуры данных. Все их многообразие подразделяется на:

- литералы и переменные;
- массивы, функции и объекты.

При этом все они делятся на встроенные и определяемые программистом. Остановимся на литералах, переменных и массивах.

Литералом называют данные, которые используются в программе непосредственно. При этом под данными понимаются числа или строки текста. Все они рассматриваются в JavaScript как элементарные типы данных. Приведем примеры литералов:

```
числовой литерал: 10
числовой литерал: 2.310
числовой литерал: 2.3e+2
строковый литерал: 'Это строковый литерал'
строковый литерал: "Это строковый литерал"
```

Литералы используются в операциях присваивания значений переменным или в операциях сравнения:

```
var a=10;
var str = 'Строка';
if(x=='test') window.alert(x);
```

Два варианта строковых литералов необходимы для того, чтобы использовать вложенные строковые литералы. Вообще говоря, есть подозрение, что равноправие "..." и '...' мнимое. Если внимательно посмотреть на реализацию страниц программирования гипертекстовых ссылок (`href.htm`, `path.htm` и `mouse.htm`), можно заметить, что вместо прямого переназначения гипертекстовой ссылки литералом типа '...' там используется косвенное переназначение через функцию литералом "...":

```
...
function line(a)
{
...
window.document.main.document.links[4].href="javascript:data(0);void(0);";
...
}
...
<A HREF="javascript:line(0);void(0);">
<IMG SRC=image.gif BORDER=0>
</A>
```

вместо:

```
<A HREF="javascript:
window.document.main.document.links[4].href='javascript:data(0);void(0);'; void(0);">
<IMG SRC=image.gif BORDER=0>
</A>
```

Это связано с особенностями реализации Netscape. Дело в том, что прямое переназначение неправильно отображает кириллицу в win32, а вот косвенное работает. Похоже, что "..." разрешает анализ информации внутри строкового литерала JavaScript-интерпретатором, а '...' – нет.

Если быть более точным, то следует сказать, что строка – это объект. У этого объекта существует великое множество методов. Строчный литерал и строчный объект – далеко не одно и то же. При применении к строчным литералам методов строчных объектов происходит преобразование первых в последние.

Переменные в JavaScript могут быть определены назначением или при помощи оператора `var`:

```
i=10;
var i;
```

```
var i=10;
var id = window.open();
var a = new Array();
```

Как видно из примеров, переменные могут принимать самые разные значения, при этом тип переменной определяется контекстом.

Переменная является свойством окна. Например, мы можем открыть окно, определить в нем новую переменную и использовать ее:

```
wid = window.open("","test","width=200,height=100,statusbar");
wid.document.open();
wid.document.write("<HTML><HEAD>");
wid.document.write("<SCRIPT>var t;</SCRIPT>");
wid.document.write("</HEAD><BODY>");
wid.document.write("<CENTER>Новое окно<BR>");
wid.document.write("<FORM>");
wid.document.write("<INPUT TYPE=button VALUE='Закрыть окно'
onClick=window.close();></FORM>");
wid.document.write("</CENTER></BODY></HTML>");
wid.document.close();
...
<A HREF="javascript:wid.t=window.prompt("Новое состояние:");wid. defaultStatus=t;wid.focus();void(0);>...</A>
```

Может ли одна и та же переменная принимать значения разных типов? Для ответа на этот вопрос рассмотрим следующий пример:

```
var flag=0;
var cid=null;
function clock()
{
flag=1;
d = new Date();
window.document.main.document.f0.fi1.value=
d.getHours()+":"+d.getMinutes()+":"+d.getSeconds();
cid = setTimeout("clock();",100);
}
function stop()
{
if(cid!=null)
{
clearTimeout(cid);
cid=null;
flag=0;
}
}
function wo()
{
cid = window.open("","test","width=400,height=100");
cid.document.open();
cid.document.write("<HTML><HEAD></HEAD><BODY><CENTER>");
cid.document.write("<FORM><INPUT TYPE=button
onClick='window.close();' value='Закрыть окно'></FORM></CENTER>");
cid.document.write("</BODY></HTML>");
cid.document.close();
cid.focus();
}
...
<FORM NAME=f0>
<INPUT NAME=fi1 SIZE=8 MAXLENGTH=8>
<INPUT TYPE=button VALUE="Часы(start/stop)" onClick="if(flag=  0)clock();else stop();">
<INPUT TYPE=button VALUE="Окно" onClick="wo();">
</FORM>
```

Можно в любом порядке нажимать на кнопки формы, и все будет работать правильно. При этом переменная cid используется и как идентификатор потока, и как идентификатор окна. Это означает, что JavaScript поддерживает полиморфизм, т.е. существуют два разных объекта с одинаковыми именами.

JavaScript позволяет во всех возможных случаях проводить преобразование одних типов данных в другие. Например, в программе используется оператор:

```
document.write("Общая сумма:" + total);
```

Если переменная total имеет значение 40, то на экране отобразится следующая строка:

Общая сумма: 40

Поскольку функция document.write управляет строковым типом данных, любые нетекстовые значения (в данном примере total) преобразуются в текстовые. И только после этого результат выводится на экран.

Этот метод вывода данных на экран применяется также для числовых значений с плавающей точкой и булевых переменных. Но существуют некоторые ситуации, в которых он не используется. Например, следующее выражение выполняется правильно в случае, если переменная total имеет значение 40:

average = total / 3;

Если же переменная total будет иметь строковый тип данных, то выполнение этого оператора приведет к возникновению ошибки.

В некоторых ситуациях строковый тип данных определен числу и необходимо преобразовать его в числовой. Для этих целей в JavaScript используются две функции:

- parseInt() – преобразует текстовый тип данных в целочисленный;
- parseFloat() – преобразует текстовый тип данных в числовой с плавающей точкой.

Обе функции считывают число в виде текста и преобразуют его в числовой тип данных. Например, вам необходимо преобразовать предложение "20 кошек" в числовое значение:

stringvar = "20 кошек";

numvar = parseInt(stringvar);

После выполнения этих операторов переменная numvar принимает значение 20. Нечисловая часть предложения игнорируется и отбрасывается.

Функции преобразования типов данных ищут числа только в начале строки текста. Если число не найдено, функция возвращает строковое значение NaN, указывая на то, что текст не содержит числовых значений.

4.10. МАССИВЫ

Массив – это упорядоченный набор элементов, содержащих значения, сохраненный под одним именем. Например, массив scores может использоваться для сохранения счета сыгранных матчей. Массивы могут состоять из чисел, строковых переменных, объектов и других типов данных.

В отличие от большинства используемых в JavaScript типов данных, массивы необходимо объявлять перед использованием.

Чтобы определить значения массива, требуется указать его индекс в скобках. Индексирование элементов массива начинается с 0, поэтому элементы объявленного выше массива имеют индексы 0 – 29. Следующие операторы определяют значения первых четырех элементов массива:

```
scores[0] = 39;
scores[1] = 40;
scores[2] = 100;
scores[3] = 49;
```

Содержание массива определяется значениями его элементов. При управлении элементами массива используются те же методы, что и при управлении значениями и переменными. Например, следующий оператор позволяет отобразить значения первых четырех элементов массива scores:

```
scoredisplay = "Статистика: " + scores[0] + "," + scores[1] + "," + scores[2] + "," + scores[3];
document.write(scoredisplay);
```

Массивы делятся на встроенные (`document.links[], document.images[],...`) и определяемые пользователем (автором документа). Для массивов задано несколько методов:

- `join();`
- `reverse();`
- `sort();`

и свойство `length`, которое позволяет получить число элементов массива.

Для определения массива пользователя существует специальный конструктор:

```
a = new Array();
b = new Array(10);
c = new Array(10,"Это значение");
```

Пример использования:

```
<SCRIPT>
c = new Array(30,"Это значение");
</SCRIPT>
<FORM><INPUT SIZE=& {c[0];};
value=& {c[1];};
onFocus="this.blur();">
</FORM>
```

Как видно из этого примера, массив может состоять из разнородных элементов. Массивы не могут быть многомерными.

4.10.1. Метод join()

Метод `join()` позволяет объединить элементы массива в одну строку. Он является обратной функцией методу `split()`, который применяется к объектам типа `STRING`. Рассмотрим пример преобразования локального URL в URL схемы `http`:

window.location:
http://tstu.ru/help/index.html
Выполнили:
b = window.location.href.split('/');
Получили массив `b`:
b[0]=http:
b[1]=
b[2]=intuit.ru
b[3]=help
b[4]=index.html
Заменили схему и вставили "host:port":

```
for(i=0;i<b.length;i++)
{
if(b[i]=="file:") b[i]="http:/";
if(b[i]=="c%7C") b[i]="remote.host.domain:80";
}
```

Получили массив `b`:
b[0]=http:
b[1]=
b[2]=intuit.ru
b[3]=help
b[4]=index.html
Слили элементы массива `b`:
l=b.join("/");
Получили в результате:
http://tstu.ru/help/index.html
Другой пример использования метода `join()` – замена символа в строке:

```
str = "document.img1.src='http://images/image1.gif';"
document.write(str);
```

Исходная строка:
document.img1.src='http://images/image1.gif';
Заменяем в строке все единицы на двойки:

```
b = str.split('1');
str = b.join('2');
```

Получаем следующий результат:
document.img2.src='http://images/image2.gif';
Последний пример показывает, что массив пользователя можно получить и без явного применения конструктора массива. Массив элементов строки получается просто как результат действия функции `split()`.

4.10.2. Метод reverse()

Метод `reverse()` применяется для изменения на противоположный порядка элементов массива внутри массива. Предположим, массив натуральных чисел упорядочен по возрастанию:

```
a = new Array(1,2,3,4,5);
```

Упорядочим его по убыванию:

```
a.reverse();
```

a[0]=5
a[1]=4
a[2]=3
a[3]=2
a[4]=1

4.10.3. Метод sort()

Как принято в современных интерпретируемых языках, например в Perl, метод `sort()` позволяет отсортировать элементы массива в соответствии с некоторой функцией сортировки, чье имя используется в качестве аргумента метода:

```
a = new Array(1,6,9,9,3,5);
function g(a,b)
{
if(a > b) return 1;
if(a < b) return -1;
```

```
if(a==b) return 0;
}
b = a.sort(g);
```

В результате выполнения этого кода получим массив следующего вида:

```
b[0]=1
b[1]=3
b[2]=5
b[3]=6
b[4]=9
b[5]=9
```

Возможность использования произвольной функции сортировки позволяет выполнять подробный анализ строковых объектов. Одним из таких примеров может служить анализ строки атрибута `SRC` контейнера `IMG`, если картинка подставляется скриптом и сортировка полей формы по значениям:

```
document.image.src = "http://www.tstu.ru:80/cgi-bin/image?x = 10&y = 20&z = 15";
```

Выделим `x` и `y`. Затем отсортируем их:

```
str = "http://www.tstu.ru:80/cgi-bin/ image?x=10&y=20&z=15";
s = str.split('?');
s1 = s[1].split('&');
s2 = s1.sort(v);
for(i=0;i<s2.length;i++)
  document.write("s2["+i+"]=
  "'+s2[i]+"'<br>");
```

```
s2[0]='x=10'
s2[1]='z=15'
s2[2]='y=20'
```

Аналогичные манипуляции можно проделать с любым массивом. Если не указывать функции в аргументе метода сортировки, то элементы массива сортируются в лексикографическом порядке. Это значит, что они сначала преобразуются в строки и только потом сортируются.

Свойство length определяет количество элементов, из которых состоит массив. Оно же определяется при создании массива. В следующем примере отображается число элементов массива scores:

```
document.write(scores.length);
```

4.11. ФУНКЦИИ: ОПИСАНИЕ И ИСПОЛЬЗОВАНИЕ

Язык программирования не может обойтись без механизма многократного использования кода программы. Такой механизм обеспечивается процедурами или функциями. В JavaScript функция выступает в качестве одного из основных типов данных. Одновременно с этим в JavaScript определен объект `Function`.

В общем случае любой объект JavaScript определяется через функцию. Для создания объекта используется конструктор, который в свою очередь вводится через `Function`.

4.11.1. Синтаксис

Перед тем как использовать функцию, необходимо ее определить. Определяют функцию при помощи ключевого слова `function`:

```
function f_name(arg1,arg2,...)
{
/* function body */
}
```

Здесь следует обратить внимание на следующие моменты. Во-первых, `function` определяет переменную `f_name`. Эта переменная имеет тип "function":

```
document.write("Тип переменной f_name:"+ typeof(f_name));
```

Тип переменной `f_name`: `function`. Во-вторых, этой переменной присваивается значение:

```
document.write("Значение i:"+i.valueOf());
document.write("Значение f_name:"+ f_name.valueOf());
```

Значение переменной `f_name`: 10. Значение переменной `f_name:function f_name(a) { if(a>=0) return true; else return false; }`. В данном случае метод `valueOf()` применяется как к числовой переменной `i`, так и к `f_name`. По этой причине функции можно назначить синоним путем присваивания ее значения другой переменной:

```
function f_name(a)
{
if(a>=0) return true; else return false;
}
document.write("Значение переменной f_name:"+f_name(1)+"");
b = f_name;
document.write("Значение переменной b:"+b(1)+"");
```

Значение переменной f_name:true

Значение переменной b:true

Традиционно функция в коде документа HTML определяется в области заголовка. Поскольку область заголовка выполняется самой первой в программе, функция определяется до ее использования.

Для того чтобы использовать функцию в программе, необходимо ее вызвать. Для вызова функции в качестве оператора необходимо указать ее имя. В скобках после названия функции указываются параметры и значения.

Следующий пример содержит как определение функции, так и ее вызов в теле страницы, чтобы продемонстрировать применяемость функции, она вызывается два раза, отображая сообщения двум разным пользователям (Васе и Клавдии).

```
<HTML>
<HEAD>
<TITLE>Функции</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function Greet(who)    {
     alert("Внимание!" + who);
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Пример функции</H1>
<P>Сообщение выводится два раза</P>
  <SCRIPT LANGUAGE="JavaScript">
Greet("Вася")
Greet("Клавдия")
</SCRIPT>
</BODY>
</HTML>
```

Наряду с функциями, отображающими на экране сообщение для пользователя, существуют функции, возвращающие в сценарий определенные значения. Это позволяет использовать функцию для вычислений. Для примера рассмотрим функцию, которая определяет среднее значение четырех чисел.

Функция начинается с ключевого слова function, названия функции и параметров. В качестве параметров будут использоваться четыре числа: a, b, c и d. Эти параметры принимают значения усредняемых чисел. Первая строка функции выглядит следующим образом:

```
function Average(a,b,c,d) {
```

Затем необходимо провести вычисление среднего значения указанных параметров:

```
result = (a+b+c+d) / 4;
```

Этот оператор создает переменную result и присваивает ей значение среднего арифметического четырех чисел. (Скобки ставятся, чтобы сделать операцию суммирования привилегированной и выполнить ее перед операцией деления.)

Для того чтобы возвратить результат в сценарий, содержащий функцию, используется ключевое слово return:

```
return result;
}
```

Полностью описание данной функции выглядит следующим образом:

```
<SCRIPT LANGUAGE="JavaScript">
function Average(a,b,c,d)  {
result = (a+b+c+d) / 4;
 return result;
}
</SCRIPT>
```

Функцию можно вызывать и как часть выражения. Например, использование оператора alert для отображения результата вычислений:

```
alert(Average(3,4,5,6));
```

Передача параметров может происходить по ссылке. В этом случае параметром функции станет объект. В следующем примере происходит вычисление площади треугольника:

```
<html>
<head>
<title>Передача параметров по ссылке</title>
<script>
<!--
function care(a, h){
  var s = a.value * h.value / 2
  document.write("Площадь треугольника равна ", s)
}
//--></script>
</head>
<body>
<form name="form1">
```

```
 Основание:  <input name="st1" type="text" size="5">
 Высота:  <input name="st2" type="text" size="5">
 <input name="" type="button" onClick="care(form1.st1, form1.st2)" value="Вычислить">
</form>
</body>
</html>
```

В качестве фактических параметров в вызове функции care выступают имена текстовых полей формы.

Использование имени формы в качестве параметра функции:

```
<html>
<head>
<title>Использование имени формы в качестве параметра функции</title>
<script>
<!--
function care(obj){
 var a = obj.st1.value
 var h = obj.st2.value
 var s = a * h / 2
 document.write("Площадь треугольника равна ", s)
}
//--></script>
</head>
<body>
<form name="form1">
 Основание:  <input name="st1" type="text" size="5">
 Высота:  <input name="st2" type="text" size="5">
 <input name="" type="button" onClick="care(form1)" value="Вычислить">
</form>
</body>
</html>
```

В предыдущих примерах вычислялось значение переменной S и для его вывода применялся метод write объекта document.

Определим в форме поле "площадь", в котором будем помещать вычисленное значение.

```
<html>
<head>
<title>Помещение вычисленного значения в поле</title>
<script>
<!--
function care(obj){
 var a = obj.st1.value
 var h = obj.st2.value
 var s = a * h / 2
 obj.res.value = s
}
//--></script>
</head>
<body>
<form name="form1">
 Основание:  <input type="text" name="st1" size="5">
 Высота:  <input type="text" name="st2" size="5">
 <input name="button" type="button" onClick="care(form1)" value="Вычислить">
 Площадь:  <input type="text" name="res" size="5">
</form>
</body>
</html>
```

Очевидно, что если функцию можно присвоить переменной, то ее можно передать и в качестве аргумента другой функции. Все это усиливается при использовании функции `eval()`, которая позволяет реализовать отложенное исполнение JavaScript-кода. Отложенное исполнение – это возможность изменения программы по ходу ее исполнения. Типичным использованием `eval()` является сокращение кода за счет генерации однотипных строк:

```
for(i=0;i<5;i++)
{
eval("document.write('test"+i+"<br>')");
}
```

Результат исполнения кода:

```
test0
```

test1
test2
test3
test4

При непосредственном кодировании пришлось бы написать пять строк кода.

4.11.2. Функция-объект

У любого типа данных JavaScript существует объектовая "обертка" – Wrapper, которая позволяет применять методы типов данных к переменным и литералам, а также получать значения их свойств. Например, длина строки символов определяется свойством `length`. Аналогичная "обертка" есть и у функций – объект `Function`.

Например, увидеть значение функции можно не только при помощи метода `valueOf()`, но и используя метод `toString()`:

```
function f_name(x,y)
{
return x-y;
}
document.write(f_name.toString()+"<br>");
```

Результат распечатки:

```
function f_name(x,y) { return x-y; }
```

Свойства функции доступны для программиста только тогда, когда они вызываются внутри функции. При этом обычно программисты имеют дело с массивом аргументов функции (`arguments[]`), его длиной (`length`), именем функции, вызвавшей данную функцию (`caller`) и прототипом (`prototype`).

Рассмотрим пример использования списка аргументов функции и его длину:

```
function my_sort()
{
a = new Array(my_sort.arguments.length);
for(i=0;i<my_sort.arguments.length;i++)
a[i] = my_sort.arguments[i];
b = a.sort();
return b;
}
b = my_sort(9,5,7,3,2)
for(i=0;i<b.length;i++)
document.write("b["+i+"]="+b[i]+"<br>");
```

Результат исполнения:

```
b[0]=2
b[1]=3
b[2]=5
b[3]=7
b[4]=9
```

Если функция может быть вызвана из других функций, то в этом случае используется свойство `caller`:

```
function slave()
{
document.write(slave.caller+"");
return slave.caller;
}
function master1()
{
slave();
}
function master2()
{
slave();
}
...
master1();
master2();
```

Результат исполнения двух последних строк:

```
function master1() { slave(); }
function master2() { slave(); }
```

Еще одним свойством объекта `Function` является `prototype`, но это общее свойство всех объектов, поэтому и обсуждать его мы будем в контексте типа данных `Object`. Упомянем только о конструкторе объекта `Function`:

```
f = new Function(arg_1,...,arg_n, body)
```

Здесь `f` – это объект класса `Function`. Его можно использовать и как обычную функцию. Конструктор используют для получения безымянных функций, которые назначают или переопределяют методы объектов. Здесь мы вплотную подошли к вопросу конструирования объектов. Дело в том, что переменные внутри функции можно рассматривать в качестве ее свойств, а функции – в качестве методов:

```
function Rectangle(a,b,c,d)
{
this.x0 = a;
this.y0 = b;
this.x1 = c;
this.y1 = d;
this.area = new Function("return Math.abs(this.x0-this.x1)* Math.abs (this.y0-this.y1)");
this.perimeter = new Function( "return (Math.abs(this.x0-this.x1)+ Math.abs(this.y0-this.y1))*2");
}
c = new Rectangle(0,0,100,100);
document.write(c.area());
```

Результат исполнения:

10000

Обратите внимание еще на одну особенность – ключевое слово `this`. Оно позволяет сослаться на текущий объект, в рамках которого происходит исполнение JavaScript-кода. В данном случае это объект с класса `Rectangle`.

4.12. ОБРАБОТЧИКИ СОБЫТИЙ

Событие – это действие, которое произвел пользователь (щелчок на ссылке или кнопке, наведение указателя на какой-либо объект, щелчок в текстовом поле перед началом его заполнения).

Обработчики событий – подпрограммы, которые позволяют программисту отслеживать действия пользователя (интерпретируемые как события) по отношению к отображенной в окне браузера html странице.

События, связанные с "мышью"

Событие	Описание
OnClick	Щелчок мышью на элементе
OnDblClick	Двойной щелчок мышью на элементе
OnMouseDown	Кнопка мыши нажата
OnMouseUp	Кнопка мыши отпущена
OnMouseMove	Указатель мыши перемещен в область элемента
OnMouseOver	Указатель мыши расположен над элементом
OnMouseOut	Указатель мыши перемещен за границы области элемента

События, связанные с клавиатурой

Событие	Описание
OnKeyDown	Клавиша нажата
OnKeyUp	Клавиша отпущена
OnKeyPress	Клавиша нажата и отпущена

События, связанные с выбором элементов и редактированием форм

Событие	Описание
OnFocus	Элемент выбран (получен фокус)
OnSelect	Часть текста внутри элемента выделена
OnChange	Данные в элементе были изменены
OnBlur	Элемент перестал быть выбранным (потерян фокус)

Ниже приведены примеры различного использования обработчиков событий.

1. Оператор присваивания в качестве значения параметра обработки событий.

```
<html>
<head>
<title>Оператор присваивания</title>
```

```
</head>
<body>
<form name="form1">
 Основание:  <input type="text" name="st1" size="5">
 Высота:  <input type="text" name="st2" size="5">
 <input type="button" value="Вычислить" onClick="form1.res.value = form1.st1.value * form1.st2.value / 2">
 Площадь:  <input type="text" name="res" size="5">
</form>
</body>
</html>
```

2. Вычисление среднего дохода.

Вводится информация о доходах за каждый месяц с первого полугодия. Определить средний доход в месяц за рассматриваемый период.

```
<html>
<head>
<title>Вычисление среднего дохода</title>
<script>
<!--
function val(obj){
 var a1 = 1 * obj.num1.value
 var a2 = 1 * obj.num2.value
 var a3 = 1 * obj.num3.value
 var a4 = 1 * obj.num4.value
 var a5 = 1 * obj.num5.value
 var a6 = 1 * obj.num6.value
 var s = (a1 + a2 + a3 + a4 + a5 + a6) / 6
 obj.res.value = s
}
//--></script>
</head>
<body>
<form name="form1">
 Январь:  <input type="text" name="num1" size="8">
 Февраль:  <input type="text" name="num2" size="8">
 Март:  <input type="text" name="num3" size="8">
 Апрель:  <input type="text" name="num4" size="8">
 Май:  <input type="text" name="num5" size="8">
 Июнь:  <input type="text" name="num6" size="8">
 <input type="button" value="Вычислить" onClick="val(form1)">
 <input type="reset" value="Сбросить">
 Средняя зарплата:  <input type="text" name="res" size="8">
</form>
</body>
</html>
```

Обработка события onClick выполняется по щелчку мыши на кнопке "Вычислить".

3. Реакция на событие onChange.

Вычислить площадь квадрата. В форме определяем 2 текстовых поля: одно для длины стороны квадрата, другое – для вычисленной площади.

Площадь квадрата вычисляется, когда значение элемента формы с именем num1 изменилось и элемент потерял фокус.

```
<html>
<head>
<title>Реакция на событие onChange</title>
<script>
<!--
function srec(obj){
 obj.res.value = obj.num1.value * obj.num1.value
}
//--></script>
</head>
<body>
</body>
<form name="form1">
 Сторона:  <input type="text" name="num1" size="8" onChange="srec(form1)">
 Площадь:  <input type="text" name="res" size="8">
 <input type="reset" value="Обновить">
```

```
</form>
</html>
```

4. Обработка события onFocus.

Площадь квадрата должна вычисляться в тот момент, когда пользователь переходит к элементу формы с помощью клавиши Tab или щелчка мыши.

```
<html>
<head>
<title>Обработка события onFocus</title>
<script>
<!--
function srec(obj){
  obj.res.value = obj.num1.value * obj.num1.value
}
--></script>
</head>
<body>
<form name="form1">
  Сторона:        <input type="text" name="num1" size="8" onFocus="srec(form1)">
  Площадь:        <input type="text" name="res" size="8">
  <input type="reset" value="Обновить">
</form>
</body>
</html>
```

5. Обработка события onBlur.

Площадь должна вычисляться в тот момент, когда элемент формы теряет фокус.

```
<html>
<head>
<title>Обработка события onBlur</title>
<script>
<!--
function srec(obj){
  obj.res.value = obj.num1.value * obj.num1.value
}
--></script>
</head>
<body>
<form name="form1">
  Сторона: <input type="text" name="num1" size="8" onBlur="srec(form1)">
  Площадь:        <input type="text" name="res" size="8">
  <input type="reset" value="Обновить">
</form>
</body>
</html>
```

6. Обработка события onSelect.

Площадь должна вычисляться в тот момент, когда выбирается часть или весь текст в текстовом поле.

```
<html>
<head>
<title>Обработка события onSelect</title>
<script>
<!--
function srec(obj){
  obj.res.value = obj.num1.value * obj.num1.value
}
--></script>
</head>
<body>
<form name="form1">
  Сторона:        <input type="text" name="num1" size="8" onSelect="srec(form1)">
  Площадь:        <input type="text" name="res" size="8">
  <input type="reset" value="Обновить">
</form>
</body>
</html>
```

7. Перестановка двух изображений.

При нажатии на кнопку "Обменять" изображения меняются местами.

```
<html>
<head>
<title>Перестановка двух изображений</title>
<script>
<!--
function chpict(){
  var l = document.pm1.src
  document.pm1.src = document.pm2.src
  document.pm2.src = l
}
//--></script>
</head>
<body>
<img src="m1.gif" name="pm1">
<img src="m2.gif" name="pm2">
<form name="form1">
  <input type="button" value="Обновить" onClick="chpict()">
</form>
</body>
</html>
```

4.13. ОРГАНИЗАЦИЯ ВЕТВЛЕНИЙ В ПРОГРАММАХ. УСЛОВНЫЙ ОПЕРАТОР

Для организации ветвлений используют условный оператор, имеющий вид:

```
if b {S1}
else {S2},
```

где b – выражение логического типа; S1, S2 – операторы.

Пример стандартного условного оператора:

```
if (a= =1) window.alert("1 найдено!");
```

Выражения условия содержат два сравниваемых значения (переменная a и число 1). В качестве значения может выступать переменная, константа или целое выражение.

Между двумя сравниваемыми значениями вводится условный оператор. Этот оператор задает условие, которому должны удовлетворять оба значения. Ниже приведены все используемые в JavaScript условные операторы.

- = = (равно)
- != (не равно)
- < (меньше)
- > (больше)
- <= (меньше или равно)
- >= (больше или равно)

Пример: максимальное значение.

```
function maxval(obj){
  var a = number (obj.num1.value);
  var b = number (obj.num2.value);
  var c = number (obj.num3.value);
  var m = a;
  if (b > m) m = b
  if (c > m) m = c
  obj.res.value = m
}
```

Поиск максимального значения с использованием объекта Math:

```
function maxval(obj){
  var a = number (obj.num1.value);
  var b = number (obj.num2.value);
  var c = number (obj.num3.value);
  obj.res.value = Math.max (Math.max (a, b), c)
}
```

Во многих случаях необходимо одновременно проверить одну переменную на выполнение нескольких условий или несколько переменных. Для этого в JavaScript имеются логические, или булевы операторы.

В приведенном ниже примере проверяются разные условия и выполняется одно и то же действие:

```
if (phone = = " ") window.alert("Ошибка!");
if (email = = " ") window.alert ("Ошибка!");
```

С помощью логического оператора эти два оператора объединяются в один:

```
if (phone == " " || email == " ") window.alert("Ошибка!");
```

В этом выражении использован логический оператор ИЛИ (||).

Оператор И (&&). Например:

if (phone == " " && email == " ") window.alert("Ошибка!");

В этом выражении вместо оператора || использован оператор &&. В этом случае сообщение об ошибке будет отображаться на экране только в том случае, если выполняются оба условия (т.е. и телефонный номер, и адрес электронной почты содержат пробелы).

В случае использования оператора && сначала проверяется первое условие. Если оно не выполняется, то второе условие проверяться не будет.

Третий логический оператор – это оператор НЕ (!). Он используется для инвертирования выражения, другими словами, в случае невыполняемости условия будут выполняться указанные операторы действия. Ниже приведен пример оператора НЕ:

if (phone != " ") window.alert("Правильно!");

В этом случае оператор ! используется как часть оператора сравнения (!=). Этот оператор инвертирует условие. Поэтому, если телефонный номер не содержит пробела, то выводится сообщение о правильности его введения.

Дополнительный оператор, используемый вместе с оператором if, – это оператор else. Этот оператор определяет действия, которые выполняются в случае невыполнения условия.

П р и м е р использования операторов if и else :

```
if (a= =1) {
window.alert("1 найдено!");
a = 0;
}
else window.alert("Неправильное значение: " + a);
```

Если условие выполняется, отображается сообщение и переменной a назначается новое значение 0. Если же условие не выполняется (если a не равно 1), то отображается другое сообщение. Если в операторе else используется несколько операторов, то необходимо заключать их в фигурные скобки.

Еще несколько примеров использования условных операторов.

1. Циклическая смена изображения.

Событие load возникает в тот момент, когда браузер начинает загрузку окна. Метод setTimeout выполняет действие, задаваемое первым параметром, по истечении указанного в миллисекундах промежутка времени, определенного вторым параметром. В качестве первого параметра задается функция **succpict()**, чем обеспечивается повторение вызова функции через каждые 2 с.

```
var k = 1
function ref()
{k = 5}
function succpict(){
  var d = document
  if (k <= 4)
    {if (k == 1)
      {d.mypict.src = "m1.gif"; k++}
    else
      if (k == 2)
      {d.mypict.src = "m2.gif"; k++}
    else
      if (k == 3)
      {d.mypict.src = "m3.gif"; k++}
    else
      if (k == 4)
      {d.mypict.src = "m4.gif"; k = 1}
    setTimeout ("succpict()", 2000)
    }
}
<body onload="succpict()">
<p>Просмотр рисунков</p>
<img src="m1.gif" name=mypict width=100>
<form name="form1">
<input type="reset" value=Остановить onClick=ref()>
<input type="button" value="Начать снова" onClick="k=1; succpict()">
</form>
</body>
</html>
```

2. Смена изображений при наведении указателя мыши.

Во время работы этого сценария смена рисунков происходит при наведении курсора мыши на изображение. При этом будет подключаться функция, которая определяет, какое изображение следует поместить в документ. Загружаемые изображения хранятся в файлах с именами m1/2/3/4.gif.

Для загрузки k-го изображения формируется имя файла по формуле "m"+k+".gif".

```
var k = 1
function succpict(){
  var d = document
  if (k < 4)
    k = k + 1
    else k = 1
    d.mypict.src = "m" + k + ".gif"
}
<body>
<h4>Для смены изображения наведите курсор</h4>
<img src="m1.gif" name=mypict width=150 onMouseOver="succpict()">
</body>
```

3. Эффект визуального удаления/приближения изображения.

Во время работы сценария, при наведении курсора мыши на изображение, оно начинает удаляться от зрителя, уменьшаясь в размерах, либо увеличиваться, моделируя эффект приближения изображения. Здесь манипулируем свойством width объекта img. При каждом вызове функций изменяется размер выводимого изображения. Функция циклически повторяется через вызов ее каждые полсекунды.

Отдаление:

```
function succpict(){
  var d = document
  var w = d.mypict.src.width
  if (w > 150)
  {d.mypict.width = w – 10
  d.mypict.src = "msm.jpg"
  setTimeout("succpict()", 500)
}
<body>
<h4>Наведите курсор на изображение</h4>
<img src="msm.jpg" name=mypict width=300 onMouseOver="succpict()">
</body>
```

Приближение:

```
function succpict(){
  var d = document
  var w = d.mypict.src.width
  if (w < 300)
  {d.mypict.width = w + 10
  d.mypict.src = "msm.jpg"
  setTimeout("succpict()", 500)
}
<body>
<h4>Наведите курсор на изображение</h4>
<img src="msm.jpg" name=mypict width=100 onMouseOver="succpict()">
</body>
```

В дополнение к оператору if язык JavaScript оснащен дополнительной возможностью построения условных выражений. Условное выражение имеет следующую конструкцию:

Переменная = (условие) ? если выполняется : если не выполняется;

Это выражение позволяет определить переменной одно из двух значений. Одно – в случае выполнения условия, второе – при его невыполняемости. Приведем пример его использования:

value = (a == 1) ? 1 : 0;

Другими словами, значение после вопросительного знака будет определяться переменной в том случае, если условие выполняется. Если условие не выполняется, то переменной определяется значение, введенное после двоеточия. Двоеточие в этом выражении играет роль оператора else, который используется не во всех случаях.

Это сокращенное выражение предпочтительней использовать вместо условных операторов при определении значения переменной. Ниже приведен пример отображения значения переменной counter:

document.write("Найдено" + counter + (counter == 1) ? " слово." : " слова."));

Если переменная counter имеет значение 1, то на экран выводится сообщение **Найдено 1 слово**. Если counter имеет значение 2 и больше, то на экран выводится сообщение **Найдено 2 слова**.

4.14. ОПЕРАТОР SWITCH И ЕГО СВОЙСТВА

Синтаксис оператора:

```
switch (B)
{ case L1: S1;
  case L2: S2; break;
  ……
  case Ln: Sn;
  default: S
}
```

B – вычисляемое выражение;
L1, L2, …, Ln – литералы;
S1, S2, …, Sn – операторы;

Каждая из строк, которая начинается оператором case, содержит значение, сравниваемое с исходным (выражение B). Если значения совпадают, выполняется оператор, указанный после ключевого слова case. В противном случае сравнивается следующее значение case.

Ключевое слово break используется для определения конца действия текущего оператора case.

Необязательный оператор default выполняется по умолчанию, если ни один из операторов case не содержит правильного значения.

После каждого оператора case можно использовать несколько операторов действий. Заключать их в скобки нет необходимости. Если условие текущего оператора case справедливо, то выполняются все операторы, введенные до ключевого слова break.

В качестве примера использования оператора switch приведем программу страницы, которая, в зависимости от введенного пользователем значения, выполняет разные действия. Пусть у пользователя запрашивается ключевое слово, определяющее страницу.

Если ключевое слово совпадает с одним из введенных в сценарии, то в окне браузера будет отображена указанная страница. Если ключевое слово не совпадает с содержащимися в сценарии, в окне браузера отобразится страница по умолчанию.

Чтобы запросить у пользователя необходимое значение, используется функция prompt(). В качестве параметра этой функции необходимо указать вопрос, отображаемый для пользователя. Вот как приблизительно это должно выглядеть:

```
where = prompt("Куда заглянем сегодня?");
```

Далее необходимо ввести оператор switch, определяющий различные варианты перехода:

```
switch (where) {
  case "Netscape" :
    window.location="http://www.netscape.com";
    break;
  case "Microsoft" :
    window.location="http://www.microsoft.com";
    break;
  case "Yahoo" :
    window.location="http://www.yahoo.com";
    break;
```

Добавьте также оператор default, позволяющий отображать в окне пользователя страницу по умолчанию:

```
default :
    window.location="http://www.nbusiness.ru";
}
```

Поскольку это последний оператор в структуре switch, нет необходимости использовать после него ключевое слово break. Скобка закрывает код оператора switch.

Полный программный код документа HTML.

```
<HTML>
<HEAD><TITLE>Проверка данных</TITLE>
</HEAD>
<BODY>
<H1>Пример работы с пользователем</H1>
        Введите слово.<BR>
<SCRIPT LANGUAGE="JavaScript">
        where = prompt("Куда заглянем сегодня?");
        switch (where) {
                case "Netscape" :
                window.location="http://www.netscape.com";
                        break;
                case "Microsoft" :
                window.location="http://www.microsoft.com";
                        break;
                case "Yahoo" :
```

```
                window.location="http://www.yahoo.com";
                        break;
                default :
                window.location="http://www.nbusiness.ru";
}
</SCRIPT>
</BODY>
</HTML>
```

Еще один пример. Определим по номеру дня его название:

```
function numday(obj)
{var m = Number(obj.num1.value);
var s
switch (m)
{case 1: s="понедельник"; break;
case 2: s="вторник"; break;
.....
case 7: s="воскресенье"; break;
default: s="ошибка в номере дня"}
obj.res.value = s}
<form name="form1">
Введите номер дня: <input type="text" size=7 name="num1">
Название дня: <input type="button" value=Определить onClick="numday(form1)">
<input type="text" size=20 name="res">
<input type="reset">
</form>
```

4.15. ЦИКЛЫ

С помощью циклов выполняются повторяющиеся операции.

Циклы for. Циклы for требуют использования временной переменной (называемой счетчиком или индексом), которая определяет количество пройденных циклов, выполненных до выхода из повторяющейся структуры. В этих структурах выход из цикла реализуется до достижения счетчиком определенного значения. Простой оператор for выглядит следующим образом:

```
for (i = 1; i < 10; i++) {
```

Цикл for состоит из трех основных элементов.

Первый параметр (например i = 1) определяет счетчик и указывает его начальное значение. Этот параметр называется *начальным выражением*, поскольку в нем задается начальное значение счетчика.

Второй параметр (i < 10) – это условие, которое должно быть справедливым, чтобы цикл выполнялся. Он называется *условием* цикла.

Третий параметр (i++) – это оператор, который выполняется при каждом последовательном прохождении цикла. Он называется *выражением инкремента*, поскольку в нем задается приращение счетчика.

После определения всех параметров цикла вводится открывающая фигурная скобка, символизирующая начало тела цикла. Закрывающая фигурная скобка вводится в конце тела цикла. Все операторы, введенные в скобках, выполняются при каждом прохождении цикла.

Как и в случае оператора if, если вы в теле цикла используете только один оператор, добавлять фигурные скобки не нужно.

Простой пример использования цикла for:

```
for (i=1; i<10; i++) {
document.write("Это строка № ",i,"<BR>");
 }
```

Это пример использования цикла для отображения сообщения при каждом прохождении цикла.

Заметьте, что цикл выполняется только девять раз. Это происходит потому, что использовано условие i < 10. Когда счетчик принимает значение 10, условие становится ложным. Если необходимо выполнить цикл десять раз, измените это условие на i<=10 или i<11.

Циклы, в которых не используются счетчики, лучше задавать оператором while.

Использование циклов while. В отличие от циклов for, цикл while не требует использования счетчиков. Наоборот, они выполняются до тех пор, пока выполняется указанное условие. Тем не менее, если условие не выполняется вообще, то цикл выполняться тоже не будет.

Оператор while содержит в скобках все необходимые параметры. В фигурных скобках традиционно указываются все выполняемые в случае истинности условия операторы.

```
while(total < 10) {
n++;
total+=values[n];
}
```

В этом цикле также используется счетчик, определяющий номера элементов массива. Вместо того, чтобы прекратиться при достижении счетчиком определенного значения, цикл прерывается, когда сумма значений массива становится больше или равно 10.

Использование цикла do...while. JavaScript содержит еще один тип циклов: do...while. Этот тип циклов сильно похож на своего родителя – цикл while. Единственная разница заключается в расположении условия. Условие в цикле do...while приводится в конце кода.

```
do {
n++;
total+=values[n];
}
while(total < 10);
```

В этом программном коде условие проверяется в конце. Это означает, что этот цикл будет выполнен по меньшей мере один раз.

Как и в циклах for и while, цикл do...while не требует добавления фигурных скобок к телу цикла, если в нем введен всего один оператор.

Прерывание цикла. Для прерывания действия цикла в тело цикла после операторов действия необходимо добавить оператор break. Пример прерывания цикла оператором break :

```
 while (true) {
 n++;
if (values[n] ==1) break;
}
```

Оператор while задает бесконечный цикл. Оператор if проверяет значения элементов массива. Если среди значений находится единица, то выполнение цикла прерывается.

При обнаружении оператора break интерпретатор JavaScript прекращает выполнение остальных операторов и переходит к выполнению оператора, следующего первым после тела цикла. Оператор break используется в любом типе циклов, как в бесконечном, так и в конечном. Это позволяет создавать процедуры выхода из циклов в случае возникновения ошибок или нахождения необходимых данных.

Повысить управляемость циклами позволяет еще один оператор – continue. Это оператор, который позволяет прервать выполнение операций текущей итерации цикла и продолжить их выполнение со следующей итерации.

```
for (i=1; i<21; i++) {
if (score[i]==0) continue;
document.write("Номер студента ",i, "оценка: ", score[i], "<BR>");
}
```

В этом коде приведен пример цикла for, используемого для вывода оценок 20 студентов, данные о которых сохранены в массиве score. Оператор if используется для сравнения значения оценки с нулем. Предполагается, что оценка 0 определяет студента, который не сдавал тест. В этом случае выполнение цикла продолжается, но результат отсутствующего на тесте студента не распечатывается.

Использование цикла for...in. Цикл for...in более гибкий, нежели привычные циклы for и while. Он специально разработан для выполнения операций со свойствами объектов.

Например, объект navigator имеет свойства, описывающие параметры браузера. Для отображения свойств объекта также проще всего использовать цикл for...in:

```
for (i in navigator) {
document.write("Свойство: ", i);
document.write(" Значение: ", navigator[i], "<BR>");
}
```

Как и обычный цикл for, этот тип цикла требует использования индекса (в нашем примере это i). Каждая итерация цикла приводит к определению нового значения переменной – индекса, соответствующего другому свойству объекта. Таким образом очень удобно управлять свойствами объектов.

4.16. ОБЪЕКТ MATH И ЕГО МЕТОДЫ

В языке JavaScript определены стандартные объекты и функции, пользоваться которыми можно без предварительного описания.

Объект Math – это встроенный в JavaScript объект, содержащий математические константы и функции. В свойствах объекта Math хранятся основные математические константы, а его методы можно использовать для вызова основных математических функций.

Метод объекта	Описание метода
abs	Абсолютное значение
sin, cos, tan	Тригонометрические функции
log	Натуральный логарифм
exp	Экспонента
min, max	Наименьшее и наибольшее значения 2-х аргументов

pow	Показательная функция (возведение в степень)
sqrt	Квадратный корень

Вычисление площади и периметра треугольника, заданного длинами сторон:

```
<html>
<head>
<title>Вычисление площади и периметра треугольнмка</title>
<script>
<!--
function care(obj){
  var a = 1 * obj.st1.value
  var b = 1 * obj.st2.value
  var c = 1 * obj.st3.value
  var s, p = a + b + c
  document.write("Периметр треугольника равен ", p)
  p = p / 2
  s = Math.sqrt(p * (p – a) * (p – b) * (p – c))
 document.write("Площадь треугольника равна ", s)
}
//--</script>
</head>
<body>
<form name="form1">
  Сторона 1:       <input type="text" name="st1" size="10">
  Сторона 2:       <input type="text" name="st2" size="10">
  Сторона 3:       <input type="text" name="st3" size="10">
  <input type="button" value="Вычислить" onClick="care(form1)">
</form>
</body>
</html>
```

Три часто используемых метода объекта Math позволяют округлять десятичные дроби до целых значений.

Math.ceil() – округляет число до ближайшего большего целого.

Math.floor() – округляет число до ближайшего меньшего целого.

Math.round() – округляет число до ближайшего целого.

Все эти методы имеют только один аргумент – округляемое значение. Это не всегда удобно: иногда возникает ситуация, когда необходимо округлять не к целому числу, а к определенному десятичному знаку (например при расчете денежных величин). Следующий пример демонстрирует, как это упущение можно устранить:

```
function round(num) {
    return Math.round(num * 100) /100;
}
```

Округляемое значение сначала умножается на 100. Таким образом, число целых разрядов увеличивается на два. Затем это число округляется и делится на 100. Таким образом, получается округленное значение, имеющее два десятичных разряда.

Один из часто используемых методов объекта Math – это метод Math.random(), позволяющий генерировать случайные числа. Этот метод не требует использования дополнительных параметров. Он возвращает произвольное десятичное число в диапазоне от нуля до единицы.

Следует заметить, что обычно необходимо получить случайное число в диапазоне от 1 до некоторого значения переменной num. Это число можно получить с помощью специальной функции. Следующий пример демонстрирует, как получить произвольное число в диапазоне от 1 до указанного числа.

```
function rand(num) {
    return Math.floor(Math.random() * num) + 1;
}
```

Эта функция получает произвольное число следующим образом: указанное число умножается на случайное значение, сгенерированное методом Math.random(), и полученное число преобразуется в целое с помощью метода Math.floor(). Чтобы число могло входить и в конец диапазона, еще прибавляется единица.

4.17. ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ STRING

Строка представляет собой группу текстовых символов, сохраняемых в одной переменной и имеющих общее название.

В JavaScript текст сохраняется в объектах String. Существует два способа создания объектов String. В первом случае мы присваиваем текстовой переменной ее значение. Во втором способе используется формальный синтаксис JavaScript. В следующих двух выражениях создается одинаковая строковая переменная:

```
test = "Это тест";
```

test = new String("Это тест");

Второй оператор дает указание браузеру создать строковый объект, содержащий текст "Это тест" и определить его переменной test.

Строковой переменной значение определяется подобно другим переменным. Определить строковой переменной значение можно и после ее создания. Например, в приведенном ниже примере переменной test (уже имеющей значение) определяется новый текст:

test = "Это только тест";

Для объединения двух текстовых значений можно использовать оператор объединения (+). В дополнение к оператору объединения можно использовать и оператор +=, позволяющий добавлять текст к содержимому уже созданного строкового объекта. В приведенном ниже примере, например, к содержимому строковой переменной sentence добавляется разделитель (.):

sentence += ".";

Количество символов, содержащихся в строковой переменной, определяется свойством length объекта String, определяемым для любого строкового объекта. Чтобы использовать это свойство в сценарии, введите название объекта, а после него .length.

Например, свойство test.length определяет длину объекта test. Ниже приведен пример использования этого свойства:

test = "Это тест";

document.write(test.length);

В первом операторе переменной test определяется значение "Это тест". Второй оператор отображает длину этого значения. В нашем примере она составляет 8 символов.

Хотя переменная test и строковая, свойство test.length имеет числовой тип данных и может использоваться в математических вычислениях.

Для изменения регистра символов текста объекта String используются два метода:

- toUpperCase() – преобразует символы текста в прописные;
- toLowerCase() – преобразует символы текста в строчные.

Например, следующий оператор позволяет отображать значение строковой переменной test символами нижнего регистра:

document.write(test.toLowerCase());

Принимая во внимание, что переменная test имеет значение "Это тест", на экране появится сообщение:

это тест

При этом значение переменной test остается прежним. Этот метод создает другой вариант значения переменной – введенный только строчными символами. Если вы хотите изменить значение строковой переменной, используйте следующий оператор:

test = test.toLowerCase();

В методах toLowerCase() и toUpperCase() в скобках дополнительные параметры не используются, хотя вводить их и обязательно.

При частом использовании строковых объектов возникает необходимость разбивки их значений на отдельные значения, сохраняемые в подстроковых переменных. Для получения части строковой переменной используется метод substring(), а метод charA() применяется для возвращения отдельного ее символа.

Метод substring() возвращает часть значения строкового объекта, определенного двумя индексами, указанными в скобках. Для примера приведем оператор, который используется для отображения 4 – 6 символов значения переменной test:

document.write(test.substring(3,6));

Правила, согласно которым определяются индексы в скобках метода substring(), следующие:

- Индексирование текста начинается с 0.
- Второй индекс определяется исключительно. Поскольку шестой символ имеет индекс 5, то в скобках указывается индекс 6.
- Оба индекса указываются в произвольном порядке. В нашем примере сначала указан меньший индекс. Вариант (6,3) приведет к тому же результату.

Еще одни пример. Пусть английскому алфавиту определяется переменная alpha:

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

Следующие операторы демонстрируют использование метода substring():

alpha.substring(0,4) возвращает значение ABCD;

alpha.substring(10,12) возвращает значение KL;

alpha.substring(12,10) возвращает значение KL. Поскольку 10<12, он используется в качестве начального индекса;

alpha.substring(6,7) возвращает значение G;

alpha.substring(24,26) возвращает значение YZ;

alpha.substring(0,26) возвращает весь алфавит;

alpha.substring(6,6) возвращает нулевое значение или пустую строку. Если индексы, приведенные в скобках, одинаковы, то всегда возвращается нулевое значение.

Метод charA() используется для возвращения отдельного символа значения строкового объекта. Для его определения в скобках следует указать индекс или расположение символа. Индексирование значения строкового объекта начинается с 0. Ниже приведен пример выполнения его для объекта alpha:

alpha.charA(0) возвращает значение A;

alpha.charA(12) возвращает значение M;

alpha.charA(25) возвращает значение Z;

alpha.charA(27) возвращает пустую строку, поскольку символа с таким индексом просто не существует.

Еще одно назначение подстроковых переменных – это отыскание строковых переменных по содержащемуся в них тексту. Для этого используется метод indexOf(). Используется этот метод следующим образом: его код добавляется после названия строкового объекта и в скобках указывается текст, по которому проводится поиск. В приведенном ниже примере в значении объекта test ищется текст "текст":

location = test.indexOf("текст");

Для большинства объектов JavaScript необходимо точно соблюдать регистр символов слов, по которым проводится поиск.

Значение, которое принимает переменная location, – это диапазон индексов символов строки, которые составляют искомое слово. Первый индекс значения строковой переменной традиционно равен 0.

При необходимости позволяется добавлять в качестве условия поиска и начальный индекс, с которого проводится поиск текста. Например, следующее выражение позволяет отыскать слово "рыба" в значении строкового объекта temp, начиная с 20-го символа:

location = temp.indexOf("Рыба",19);

Предназначение второго параметра метода indexOf() – это поиск нескольких одинаковых текстовых фрагментов в значении строкового объекта. Зная расположение первого текстового фрагмента, условия поиска задаются таким образом, чтобы избежать его нахождения и выполнить поиск второго такого же фрагмента.

Второй метод, lastIndexOf(), выполняется подобным образом, но возвращает индекс последнего найденного текстового фрагмента в значении строковой переменной. Например, ниже приведен оператор отыскания индекса последнего расположения имени "Вася" в значении объекта names:

location = names.lastIndexOf("Вася");

Как и в случае с методом indexOf(), второй параметр используется для задания начального индекса поиска. Но в этом случае поиск проводится в направлении убывания индексов, а не увеличения.

В JavaScript, наряду с числовыми массивами, активно используются и строковые массивы. Это позволяет просто и эффективно управлять большими объемами текстовой информации.

Объявление строковых массивов проводится тем же способом, что и объявление числовых массивов (в JavaScript не существует принципиальной разницы между числовыми и строковыми массивами):

```
names = new Array(30);
```

Следующим образом объявляются его элементы:

```
names[0] = "Виталий";
names[1] = "Юлия";
```

Элементы строкового массива используются везде, где позволено использовать строковые переменные. Все описанные выше методы выполняются и для элементов строковых массивов. Например, следующий оператор позволяет распечатать первые пять символов значения первого элемента массива names:

```
document.write(names[0].substring(0,5));
```

JavaScript содержит метод split(), позволяющий разделять строку на составные части. Для того чтобы правильно его использовать, необходимо указать необходимый объект и символ, по которому проводится разделение:

```
test = "Виталий";
parts = test.split("и");
```

В этом примере строковая переменная test принимает значение Виталий. Метод split() разделяет значение на три составные части по символам и. После выполнения этого метода массив определен следующим образом:

```
parts[0] = "В"
parts[1] = "тал"
parts[2] = "й"
```

JavaScript обладает еще одним методом управления элементами строкового массива: join(). Он выполняет операцию, обратную методу split(). Следующий оператор объединяет элементы массива parts в одну строку:

```
fullname = parts.join("и");
```

Параметр в скобках определяет символ, по которому проводится объединение. В нашем случае в качестве этого символа выступает буква и. В результате получим исходное имя Виталий. Если символ указывать нет необходимости, обязательно введите в качестве параметра запятую.

JavaScript содержит метод sort(), используемый для сортировки элементов массива. Он возвращает упорядоченную версию исходного массива. Упорядочение проводится как по алфавиту (для строковых массивов), так и по возрастанию или убыванию (для числовых). В следующем примере упорядочен массив, состоящий из четырех английских имен:

```
names[0] = "Public, John Q.";
names[1] = "Tillman, Henry J.";
names[2] = "Clinton, Bill";
names[3] = "Mouse, Micky";
sortednames = names.sort();
```

Последний оператор создает в массиве sortednames упорядоченные по алфавиту элементы массива names.

В качестве примера практического использования строковых переменных рассмотрим сценарий создания бегущей строки.

Для начала определяется сообщение, которое будет отображаться в бегущей строке. Для сохранения текста сообщения будем использовать строковую переменную msg. В самом начале сценария определим значение переменной:

msg = "Это пример бегущего сообщения."

Далее определяем следующую строковую переменную spacer. Ее значение будет отображаться между копиями значений переменной msg:

spacer = " ";

Кроме того, нам понадобится еще одна переменная – числовая переменная, имеющая числовое значение места расположения строки. Назовем ее pos и определим начальное значение 0.

Создание бегущего сообщения проводится с помощью функции ScrollMessage(). Код функции ScrollMessage():

```
 function ScrollMessage() {
window.status = msg.substring(pos, msg.length) + spacer + msg.substring(0,pos);
 pos++;
if (pos > msg.length) pos = 0;
window.setTimeout("ScrollMessage()",200);
}
```

4.18. УПРАВЛЕНИЕ ФОКУСОМ

Фокус – это характеристика текущего окна, фрейма или поля формы. Под фокусом понимают возможность активизации свойств и методов объекта. Например, окно в фокусе, если оно является текущим, т.е. лежит поверх всех других окон и исполняются его методы или можно получить доступ к его свойствам.

4.18.1. Управление фокусом в окнах

Для управления фокусом у объекта класса "окно" существуют два метода: `focus()` и `blur()`. Первый передает фокус в окно, в то время как второй фокус из окна убирает. Рассмотрим простой пример:

```
function hide_window()
{
wid=window.open("","test",
   "width=400,height=200");
wid.opener.focus();
wid.document.open();
... wid.document.close();
}
```

В данном примере новое окно открывается и сразу теряет фокус, прячется за основным окном-родителем. Если при первичном нажатии на кнопку оно еще всплывает и только после этого прячется, то при повторном нажатии пользователь не видит появления нового окна, так как оно уже открыто и меняется только его содержимое.

Для того чтобы этого не происходило, нужно после открытия передавать фокус на новое окно:

```
function visible_window()
{
wid=window.open("","test",
   "width=400,height=200");
wid.focus();
wid.document.open();
... wid.document.close();
}
```

Если теперь нажимать попеременно кнопки "Скрытое окно" и "Видимое окно", окно будет то появляться, то исчезать. При этом новых окон не появляется, так как с одним и тем же именем может быть открыто только одно окно.

Невидимое окно может доставить пользователю неприятности, из которых самая безобидная – отсутствие реакции на его действия. Код просто записывается в невидимое окно. Но ведь в скрытом окне можно что-нибудь и запустить. Для этого стоит только проверить, существует ли данное окно или нет, и если оно есть и не в фокусе, то активизировать в нем какую-нибудь программу.

Для реализации такого сценария достаточно использовать метод окна `onblur()`. Его можно также задать в контейнере `BODY` в качестве обработчика события `onBlur`, но в этом случае он видим пользователю. Мы воспользуемся этим методом "в лоб":

```
window.onblur = new Function("window.defaultStatus =
'Background started...';");
window.onfocus = new Function("window.defaultStatus =
'Document:Done';");
```

Обратите внимание на поле статуса браузера. Оно демонстрирует возможность выполнения функции в фоновом режиме. Кроме того, `onblur()` в этом виде не отрабатывает в Internet Explorer. Причина кроется в прототипе объекта и возможности его переназначения программистом.

4.18.2. Управление фокусом во фреймах

Фрейм – это такое же окно, как и само окно браузера. Точнее – это объект того же класса. К нему применимы те же методы, что и к обычному объекту "окно":

```
var flag=1;
function clock()
{
if(flag==0)
{
d=new Date();
s=d.getHours()+':'+d.getMinutes()+':'+
  d.getSeconds();
window.document.forms[0].elements[0].value=s;
}
setTimeout('clock();',100);
}
window.onblur =
  new Function('this.flag = 1;');
window.onfocus =
  new Function('this.flag = 0;');
window.onload = clock;
```

Данный фрагмент кода размещен в каждом из двух фреймов, которые отображаются в примере. А их именно два. Просто ширина границы набора фреймов установлена в 0. Если окно примера разделить мысленно пополам и "кликнуть" мышью в одну из половин, то пойдут часы в этой половине. Если теперь переместиться в другой фрейм и "кликнуть" мышью в нем, то часы пойдут в поле формы этого фрейма, а в другом фрейме остановятся.

Контрольные вопросы

1. Что такое сценарий?
2. Каким образом сценарий JavaScript включается в тело HTML-страницы?
3. Как использовать внешний файл со сценарием?
4. Какие типы данных существуют в языке JavaScript?
5. Как определяются переменные и функции?
6. Назовите основные типы выражений JavaScript.
7. Как происходит описание функций?
8. Что такое событие? Обработчик события?
9. Какие существуют события, связанные с мышью?
10. Какие существуют события, связанные с клавиатурой?
11. Как происходят передача и обработка параметров из форм?
12. Можно ли использовать JavaScript для математических вычислений?
13. Какой оператор используют для организации ветвлений?
14. Что происходит при использовании оператора switch?
15. Какими качествами обладает программный объект?
16. Опишите иерархическую структуру объекта JavaScript.
17. Перечислите обязательные объекты страницы.

ЗАКЛЮЧЕНИ

В первой части пособия рассмотрены основные технологии, используемые при разработке клиентских приложений для Internet. Главное внимание уделялось языку разметки гипертекстовых документов HTML, его функциям, свойствам и параметрам. Сегодня применение HTML практикуется во всех без исключения электронных документах, независимо от тематики, величины и коммерческой направленности Интернет-проекта.

Однако мир меняется и современные информационные технологии тоже не стоят на месте: применение стандартных функций HTML становится недостаточным для Web-разработчика, который стремится к максимальной интерактивности на страницах своего сайта.

Поэтому в пособии были рассмотрены также технологии CSS и JavaScript как основные интерактивные скриптовые технологии, чтобы показать эффективность использования HTML. Применение Dynamic HTML, CGI и других технологий является отдельной темой, подходящей для отдельной книги.

Что касается HTML CSS и JavaScript, то можно с уверенностью утверждать, что, изучив это пособие, вы освоили азы использования этих языков и при желании способны развивать свои навыки в области разработки электронных документов.

ОГЛАВЛЕНИЕ