Федеральное агентство по образованию
Волгоградский государственный педагогический университет

А.Н.Сергеев

JavaScript

использование форм в активных документах Web

Методическая разработка

Волгоград
2005

Рецензенты:

Лецко В.А., кандидат педагогических наук

Сергеев А.Н.

JavaScript: использование форм в активных документах Web. Методическая разработка. – Волгоград, 2005.

Практическое руководство «JavaScript: использование форм в активных документах Web» рассчитан на тех, кто ведет разработку Web-документов, использующих возможности компьютерного диалога. Пособие содержит описание форм и их элементов, а также практические рекомендации по использованию сценариев JavaScript для обработки данных, указанных пользователем. Рекомендуется для студентов педагогических ВУЗов, изучающих вопросы создания интерактивных образовательных ресурсов Интернет.

Содержание

Введение

Данное пособие посвящено вопросам использования форм в активных документах Web. Формы позволяют создавать на HTML-страницах такие поля и органы управления, как кнопки, текстовые поля, переключатели, списки выбора и др. Эти элементы являются неотъемлемой частью различных программ, построенных на основе компьютерного диалога и широко используются для создания интерактивных образовательных ресурсов Интернет.

Изложение материала ведется на основе использования сценариев, предназначенных для выполнения браузером на стороне пользователя (язык JavaScript). Это означает, что для работы вам потребуется лишь браузер и простой текстовый редактор. Этих средств будет достаточно, чтобы проверить работу всех примеров, приведенных в книге, создать свои активные документы Web, использующие все возможности предложенных здесь технологий.

В первом разделе рассматриваются общие вопросы использования форм и различные подходы к обработке данных, указанных пользователем.

Самый объемный раздел – второй. В нем приводятся сведения по использованию основных полей и органов управления форм (кнопок, текстовых полей, переключателей, списков выбора), их включению в документы HTML, обработке с помощью сценариев JavaScript.

В третьем разделе рассматриваются поля и органы управления форм, которые обеспечивают дополнительные возможности для создания активных документов Web, построенных на основе как клиентских, так и серверных приложений.

Четвертый раздел посвящен использованию cookie для долговременного хранения данных, указанных пользователем с помощью активных элементов форм, а также передачи этих данных между различными документами HTML, без чего сложно представить многие диалоговые обучающие программы, которые должны отслеживать продвижение обучаемых по образовательной траектории, фиксировать их результаты на этом пути.

В пятом разделе приводятся дополнительные сведения по внешнему оформлению полей и органов управления, что позволяет сделать компьютерный диалог более дружественным и интуитивно понятным пользователю.

Материал книги излагается последовательно и многие последующие части опираются на сведения, приведенные ранее. Вместе с тем, мы рекомендуем подходить к изучению данного пособия «в два подхода». В первый подход следует последовательно просмотреть всю книгу, не вдаваясь в технические детали, но отмечая для себя ключевые моменты и возможности, которые предлагают технологии форм и сценариев JavaScript для их обработки. Это позволит получить общее и целостное представление о

возможностях технологий Интернет по созданию диалоговых программ, построенных на основе Web. Второй подход следует совершать «с задачей на руках», когда вами уже определены цели работы с формами и вы представляете свой конечный результат. Здесь данное пособие будет выступать как справочник и источник идей для технической реализации ваших задумок. Надеемся, что последнему будут способствовать примеры, приведенные в данном пособии, которые подобраны так, чтобы не только иллюстрировать очередной теоретический блок, но и служить заготовками для более сложных страниц Web, использующих формы и динамическое содержимое для создания интерактивных, диалоговых программ.

Раздел 1. Основы использования форм в документах HTML

Возможность включения форм в страницы Web подразумевается самим языком HTML. Язык HTML позволяет создать форму, определить ее элементы, их внешний вид и расположение. Вместе с тем, формы предназначены для ввода предназначенных для некоторой обработки данных, а эта задача уже требует выхода за пределы HTML, обращения к некоторым программам, которые способны получить введенные данные и на их основе сформировать ответную страницу.

Способы обработки данных форм

Можно выделить два принципиально различных способа организации программ, предназначенных для обработки данных форм. Первый способ – это использование серверных CGI-приложений. Этот способ подразумевает, что программа обработки данных форм выполняется сервером Web. Данные, введенные пользователем на Web-странице, пересылаются серверному приложению. Приложение эти данные обрабатывает и на их основе формирует новую HTML-страницу, которая пересылается обратно пользователю в качестве результата его запроса. По подобной схеме работают многие популярные Web-сервисы: поисковые машины, форумы, службы электронной почты и т.п.

Серверное приложение может долговременно сохранять данные, введенные пользователем (например, учетные записи и личные данные пользователя), обращаться к базам данных (например, к базам поисковых систем), запускать дополнительные приложения для выполнения некоторых операций с данными пользователя (например, для отсылки электронного письма, введенного пользователем на странице Web). Вместе с тем, весь диалог с пользователем строится на основе пересылки полных HTML-страниц, что не всегда удобно, не позволяет интерактивно менять элементы уже загруженных страниц, всегда связано с довольно значительными

задержками времени, обусловленными пересылкой данных и перезагрузкой Web-страниц по достаточно медленным каналам Internet.

В большинстве случаев серверные CGI-приложения создаются с помощью языка Perl, но это не обязательно. Для создания подобных программ может подойти и любой другой язык, способный работать со стандартными потоками ввода и вывода той платформы, на которой выполняется сервер Web. CGI-приложения создаются на таких языках, как C, C++, Pascal и др.

Второй способ создания приложений, способных обрабатывать данные форм, это использование языков сценариев (как правило, JavaScript), выполняемых браузером на стороне клиента. Этот способ не подразумевает пересылок данных удаленным приложениям, вся обработка выполняется браузером и сценарии способны менять не страницы целиком, а лишь их элементы. Все это означает, что второй способ подходит для создания **интерактивных** Web-страниц, способных взаимодействовать с пользователем в **диалоговом** режиме. Как недостаток этого способа следует указать отсутствие возможности долговременного и централизованного хранения данных, введенных пользователем. Этот недостаток частично снимается возможностью использования cookie, которые позволяют организовать хранение небольших по объему данных на компьютере пользователя. В полной мере решить эту проблему и обеспечить интерактивность диалоговых Web-страниц можно лишь созданием комплекса программ, совмещающих оба представленных способа работы с формами.

Создание форм в документах HTML

Для создания форм в документах HTML используется оператор (тег) `<FORM>`. Этот тег является парным и в его пределах размещаются все операторы полей и органов управления форм (кнопок, текстовых полей и т.п.). С точки зрения дизайнерского оформления страницы, эти элементы в большинстве случаев можно рассматривать как некоторые графические изображения, что позволяет определять их расположение в соответствии с принципами форматирования документов HTML. Это означает, что внутри конструкции `<FORM>...</FORM>` могут располагаться и другие конструкции HTML: абзацы, таблицы, линии, графические изображения и т.п. Внешний вид элементов форм определяется параметрами самих этих элементов, а также стилевыми параметрами, определенными с помощью каскадных таблиц стилей. Заметим также, что один документ HTML может содержать несколько форм (например, форма поиска и форма регистрации).

В наиболее общем виде оператор `<FORM>` выглядит следующим образом:

```
<FORM
  NAME="Имя_формы"
  TARGET="Имя_окна"
  ACTION="Адрес_URL_CGI-приложения"
  METHOD="GET" или "POST"
  ENCTYPE="Кодировка_данных"
  onSubmit="Обработчик_события_Submit">
    . . .
    определение полей и органов управления
    . . .
</FORM>
```

Параметр NAME задает имя формы. Это имя нужно для адресации формы как свойства объекта document, которое удобно использовать для обращения к данным форм из сценариев JavaScript.

Назначение параметра TARGET аналогично назначению этого же параметра в операторе <A>. Когда форма используется для передачи запроса CGI-приложению, ответ, полученный от сервера, отображается в окне. Имя этого окна задано параметром TARGET. Если ответ должен отображаться в том же окне, что и форма, то параметр TARGET задавать не нужно.

С помощью параметра ACTION указывается адрес URL загрузочного файла CGI-приложения, а также передаваемые ему параметры. В том случае, когда форма предназначена для передачи данных приложению сервера Web, параметр ACTION является обязательным. Но если данные, введенные в форме, обрабатываются сценарием JavaScript локально и не передаются серверу Web, этот параметр значения не имеет.

Параметр METHOD задает метод передачи данных из формы расширению сервера Web и может принимать значения GET или POST. Если данные из полей формы обрабатываются сценарием JavaScript локально, параметр METHOD задавать не нужно.

Параметр ENCTYPE задает тип MIME передаваемых данных и используется очень редко. Если форма предназначена для передачи текстовых данных (как это обычно бывает), этот параметр по умолчанию имеет значение application/x-www-form-urlencoded. В этом случае для передаваемых данных используется так называемая кодировка URL. Тип данных multipart/form-data позволяет передавать как текстовые, так и двоичные данные. При локальной обработке данных сценарием JavaScript параметр ENCTYPE не задается.

Помимо параметров, для формы можно определить обработчик события, связанный с кнопкой типа submit. Такая кнопка предназначена для посылки данных из заполненной формы расширению сервера Web. Назначив обработчик события, сценарий JavaScript может управлять этим процессом, что позволяет организовывать предварительную обработку данных сценарием JavaScript, отсылаемых приложению CGI.

Доступ к формам из сценариев JavaScript

Сценарии JavaScript могут получить доступ к формам как к свойствам объекта document. Как указано выше, к этим свойствам удобно обращаться по их названиям, которые определяются параметром NAME включенной в документ формы. Каждое такое свойство, в свою очередь, также является объектом класса form и содержит свои свойства – вложенные объекты.

Объект класса form имеет два набора свойств, состав одного из которых является фиксированным, а состав другого зависит от того, какие поля и органы управления определены в форме. Первый набор свойств практически полностью определяется параметрами тега <FORM> и приведен ниже:

Свойство	*Описание*
action	Содержит значение параметра ACTION
elements	Массив всех элементов (полей и органов управления), определенных в форме
encoding	Содержит значение параметра ENCTYPE
length	Размер массива elements
method	Содержит значение параметра METHOD
target	Содержит значение параметра TARGET

Второй набор свойств определяется набором полей и органов управления, входящих в состав формы. Ниже будет показано, что для каждого такого элемента можно указать свое имя. Это позволяет обращаться к элементу не только через массив elements, но и как к отдельному свойству формы.

Раздел 2. Основные поля и органы управления форм

Прежде чем приступить к подробному изучению полей и органов управления форм, перечислим основные из них, снабдив краткими описаниями.

Кнопки	
Button	«Обычная» кнопка с надписью
Submit	Кнопка для посылки данных из заполненной формы расширению сервера Web. Внешним видом не отличается от кнопки button
reset	Кнопка, с помощью которой пользователь может сбросить содержимое полей ввода и состояние переключателей в их

	исходное состояние. Внешним видом не отличается от кнопки button
Текстовые поля	
Text	Однострочное текстовое поле
password	Текстовое поле для ввода паролей. Набранный в этом поле текст не отображается на экране
textarea	Многострочное текстовое поле
Переключатели и списки выбора	
checkbox	Переключатель типа «флажок». Может использоваться в составе набора независимых друг от друга переключателей или отдельно
radio	Переключатель для группы зависимых друг от друга переключателей. Используется для выбора одного значения из нескольких возможных
select	Список выбора произвольных текстовых строк. Обычно используется как выпадающий список

Кнопка button

В настоящем пособии мы подробно рассмотрим использование кнопки button. Кнопки submit и reset предназначены в основном для создания форм, работающих «в паре» с серверными приложениями и нами использоваться не будут. Кроме того, существует возможность использования графических кнопок и кнопок с изображениями. О них речь пойдет в разделе «Дополнительные поля и органы управления» данного пособия.

В общем виде кнопка класса button определяется в форме с помощью оператора <INPUT> следующим образом:

```
<INPUT TYPE="button"
  NAME="Имя_кнопки"
  VALUE="Надпись_на_кнопке"
  onClick="Обработчик_события">
```

Параметр TYPE оператора <INPUT> должен иметь значение button, как это показано выше (для кнопок submit и reset – соответственно submit и reset). С помощью параметра NAME задается имя объекта, соответствующего кнопке (а не надпись на кнопке). Это имя используется как имя свойства-объекта в составе формы. Надпись на кнопке указывается с помощью параметра VALUE. От длины надписи зависит и размер кнопки. Определив обработчик события onClick, можно задать сценарий JavaScript, который получит управление после того как пользователь нажмет на кнопку.

Свойства объекта button

Объект button имеет два свойства, отражающие значения соответствующих параметров оператора <INPUT>:

Свойство	*Описание*
name	Значение параметра NAME
value	Значение параметра VALUE

Методы объекта button

Для объекта button определен всего один метод, не имеющий параметров – метод click(). Вызов этого метода приводит к такому же эффекту, что и щелчок левой клавишей мыши по кнопке.

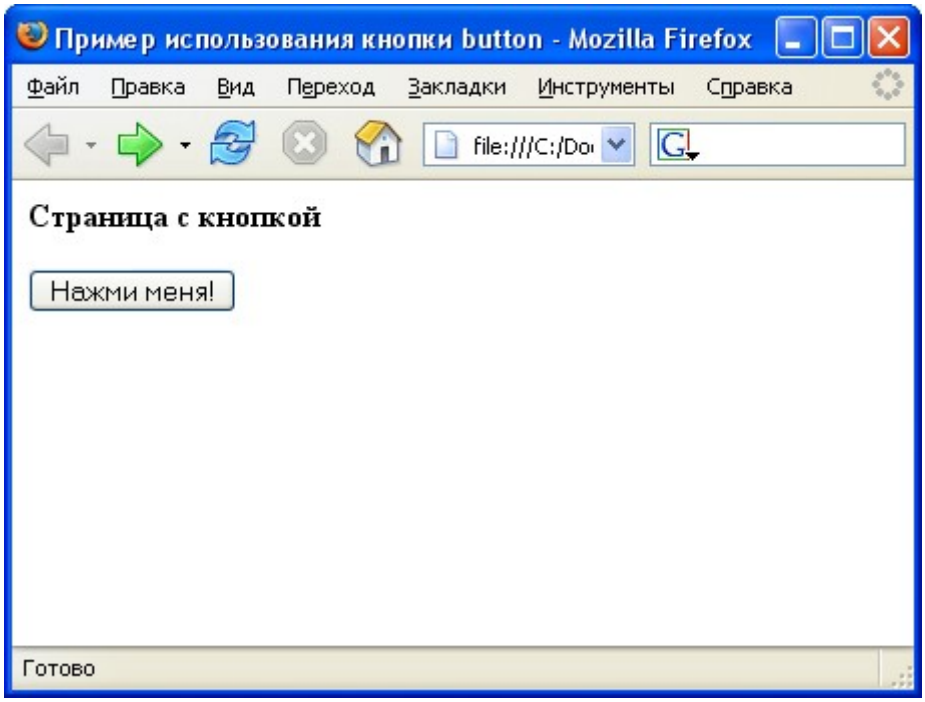
Пример 2.1. Страница с кнопкой

В качестве примера приведен сценарий, выполняющий обработку щелчка по кнопке с надписью «Нажми меня». Если нажать на эту кнопку, сценарий отображает в окне браузера сообщение с благодарностью.

```
<HTML>
  <HEAD>
    <TITLE>Пример использования кнопки button</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    function btnClick()
    {
      var szTxt="";
      szTxt=document.frm.btn.value;
      document.write("<b>Спасибо!</b><br>");
      document.write("Вы нажали кнопку: <b>&quot;" +
                     szTxt + "&quot;</b>");
    }
    // -->
    </SCRIPT>
  </HEAD>
  <BODY>
    <b>Страница с кнопкой</b>
    <FORM NAME="frm">
      <P><INPUT TYPE="button" NAME="btn"
          VALUE="Нажми меня!" onClick="btnClick();">
    </FORM>
  </BODY>
</HTML>
```

Рассмотрим данный пример подробнее. Внешний вид приведенной страницы показан на рисунке 2.1.

Рисунок 2.1. Документ HTML с кнопкой button



Для создания кнопки, в теле документа HTML определена форма с именем `frm`. Эта форма содержит всего один элемент – кнопку `button` с именем `btn` и надписью `«Нажми меня!»`:

```
<FORM NAME="frm">
  <P><INPUT TYPE="button" NAME="btn"
      VALUE="Нажми меня!" onClick="btnClick();">
</FORM>
```

Для кнопки назначен обработчик события `onClick`. Обработчик – это функция `btnClick()`. Функция написана на языке JavaScript и определена в заголовке документа в соответствии с правилами включения кода JavaScript в страницы HTML:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
function btnClick()
{
  var szTxt="";
  szTxt=document.frm.btn.value;
  document.write("<b>Спасибо!</b><br>");
  document.write("Вы нажали кнопку: <b>&quot;" +
```

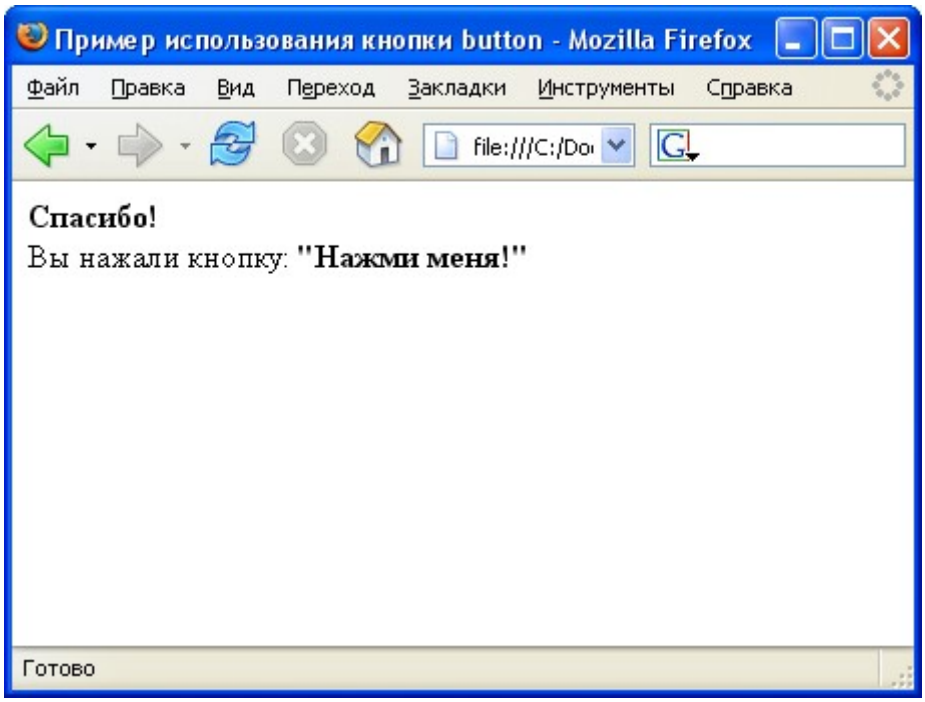
```
                    szTxt + "&quot;</b>");
}
// -->
</SCRIPT>
```

Функция `btnClick()` выводит в окно документа две строки, содержащие текст и теги HTML:

```
document.write("<b>Спасибо!</b><br>");
document.write("Вы нажали кнопку: <b>&quot;" +
                    szTxt + "&quot;</b>");
```

Результат работы функции `btnClick()` приведен на рисунке 2.2.

Рисунок 2.2. Результат работы функции btnClick()



Для подготовки сообщения, функция `btnClick()` использует переменную `szTxt`, которой присваивается текстовое значение – надпись на кнопке:

```
szTxt=document.frm.btn.value;
```

Обратите внимание, что надпись на кнопке – это свойство `value` созданного нами объекта `btn`. Кнопка (объект `btn`), в свою очередь, является объектом формы `frm`. Форма является составной частью документа (объект `document`). Таким образом получается, что доступ к имени кнопки можно получить, обратившись к конструкции `document.frm.btn.value`.

Подобный способ обращения к свойствам полей формы мы будем использовать и в дальнейшем.

Однострочное поле text

Однострочные текстовые поля предназначены для ввода и редактирования простых текстовых данных – строк поиска, анкетных данных, электронных адресов и т.п. Чтобы встроить такое поле в форму, необходимо использовать уже знакомый оператор `<INPUT>` с параметром `TYPE`, равным значению `"text"`:

```
<INPUT TYPE="text"
  NAME="Имя_текстового_поля"
  VALUE="Значение_по_умолчанию"
  SIZE="Размер_поля"
  MAXLENGTH="Максимальное_число_символов"
  onBlur="Обработчик_события"
  onChange="Обработчик_события"
  onFocus="Обработчик_события"
  onSelect="Обработчик_события">
```

Параметр `NAME` позволяет задать имя поля, необходимое для обращения к свойствам соответствующего объекта `text`.

С помощью параметра `VALUE` можно записать в поле произвольную текстовую строку, которая будет являться значением по умолчанию. Эта строка будет отображаться сразу после загрузки документа HTML в окно браузера.

Параметр `SIZE` определяет размер (ширину) видимой части текстового поля в символах и влияет на его внешнее отображение на странице HTML. Ограничить длину строки, которую можно ввести в текстовое поле, можно с помощью параметра `MAXLENGTH`. Значение этого параметра может превышать значение параметра `SIZE`, так как браузер обеспечивает прокрутку текста в текстовом поле.

Свойства объекта text

Ниже приведены три свойства, которыми обладает однострочное текстовое поле:

Свойство	*Описание*
`name`	Значение параметра `NAME`
`value`	Текущее содержимое поля редактирования (введенный текст)
`defaultValue`	Значение параметра `VALUE` (текстовая строка,

заданная по умолчанию)

Сразу после отображения поля редактирования свойства `defaultValue` и `value` хранят одинаковые строки. Когда пользователь редактирует текст, все изменения отражаются в свойстве `value`. Заметим, что изменяя содержимое свойства `value`, сценарий JavaScript может изменить содержимое поля редактирования.

Методы объекта text

Для объекта `text` определены методы `focus()`, `blur()` и `select()`, не имеющие параметров. С помощью метода `focus()` сценарий JavaScript может передать фокус ввода полю редактирования (поле становится активным, включается текстовый курсор, ожидается ввод текста), а с помощью метода `blur()` – отобрать фокус у этого поля. Вызов метода `select()` приводит к выделению содержимого поля редактирования.

Обработчики событий объекта text

Как показано выше, объект `text` способен реагировать на четыре события – `onFocus`, `onBlur`, `onChange`, `onSelect`. Ниже представлено их описание:

Событие	*Когда возникает*
`onFocus`	Возникает, когда поле редактирования получает фокус ввода
`onBlur`	Возникает, когда поле редактирования теряет фокус ввода
`onChange`	Возникает при изменении содержимого поля редактирования
`onSelect`	Возникает при выделении содержимого поля редактирования

Пример 2.2. Анкета

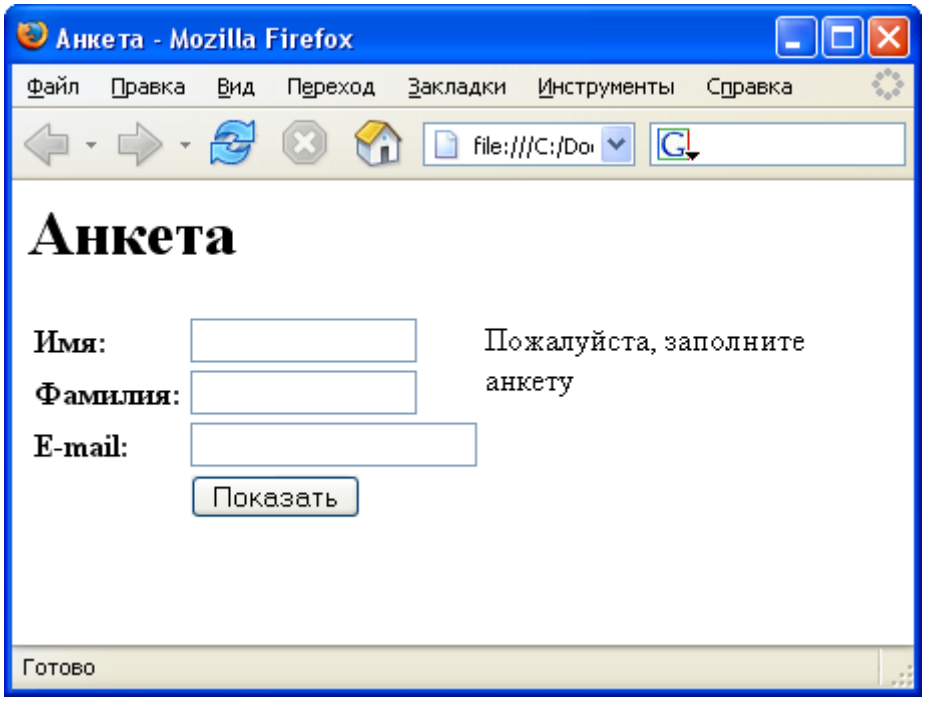
Работу с текстовыми полями рассмотрим на основе простого примера. Создадим анкету, которая запрашивает данные пользователя и выводит их в диалоговом окне.

```
<HTML>
<HEAD>
<TITLE>Анкета</TITLE>
```

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  function Complete()
  {
    var szStr="";
    szStr="Имя: " + frm.name.value +
          "\nФамилия: " + frm.family.value +
          "\nE-mail: " + frm.email.value;
    alert(szStr);
  }
// -->
</SCRIPT>
</HEAD>

<BODY>
  <H1>Анкета</H1>
  <FORM NAME="frm">
  <TABLE>
  <TR>
    <TD><B>Имя:</B></TD>
    <TD><INPUT NAME="name" TYPE="text" SIZE="15"></TD>
    <TD rowspan="4" valign="top">Пожалуйста,
                                 заполните анкету</TD>
  </TR>
  <TR>
    <TD><B>Фамилия:</B></TD>
    <TD><INPUT NAME="family" TYPE="text" SIZE="15"></TD>
  </TR>
  <TR>
    <TD><B>E-mail:</B></TD>
    <TD><INPUT NAME="email" TYPE="text" SIZE="20"></TD>
  </TR>
  <TR>
    <TD> </TD>
    <TD><INPUT VALUE="Показать" TYPE="button"
                            onClick="Complete();"></TD>
  </TR>
  </TABLE>
  </FORM>
</BODY>
</HTML>
```

Внешний вид данного документа представлен на рисунке 2.3.

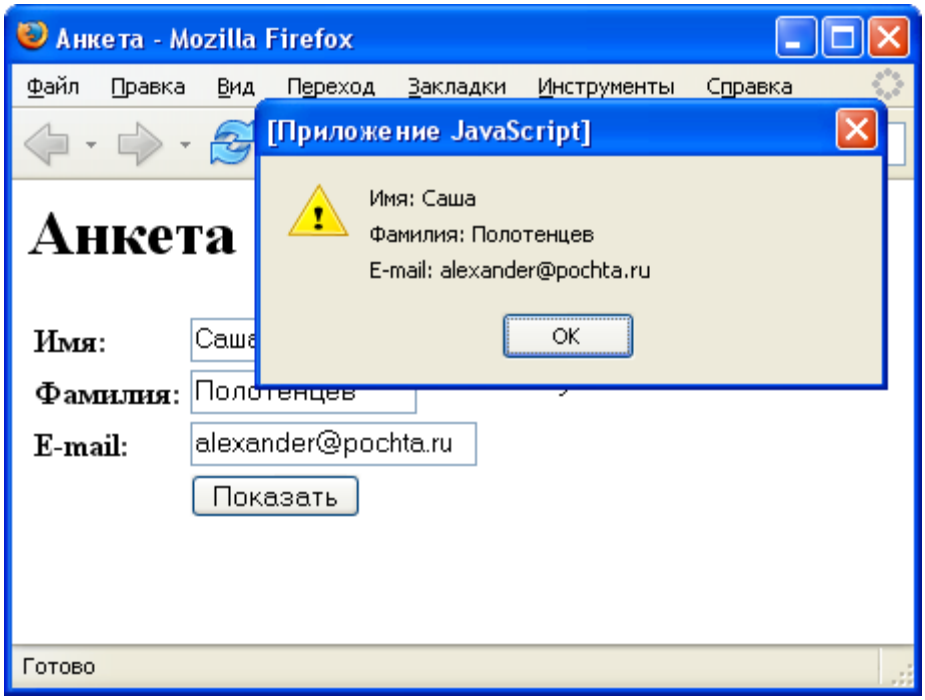
Рисунок 2.3. Внешний вид документа «Анкета»



Документ содержит форму с тремя текстовыми полями (`name`, `family`, `email`) и одной кнопкой. Для более удобного расположения элементов форм и поясняющих надписей используется таблица, состоящая из четырех строк и трех столбцов (в последнем столбце объединены все строки).

В текстовые поля вводятся необходимые данные, а при нажатии на кнопку «Показать» - выводятся в диалоговом окне (рисунок 2.4)

Рисунок 2.4. Диалоговое окно с анкетными данными



Чтобы лучше разобраться во всех деталях работы предложенного сценария, обратим внимание на следующие ключевые моменты:

1. Включенная в документ форма имеет название `frm`.
2. Текстовые поля имеют названия `name` (имя), `family` (фамилия), `email` (адрес электронной почты).
3. Кнопка «Показать» названия не имеет, но для нее назначен обработчик события `onClick`. В качестве обработчика выступает функция `Complete()`, определенная в заголовке документа HTML.
4. Функция `Complete()`, используя указанные в п.1 и 2 имена формы и ее элементов, обращается к значениям текстовых полей, формирует строку с анкетными данными, вызывает диалоговое окно для вывода этой строки.

Пример 2.3. Анкета с подсказками

Усовершенствуем рассмотренный выше пример. Внесем в него следующие изменения:

1. Добавим подсказки для каждого текстового поля. Подсказки будут выводиться в правой части окна анкеты всякий раз, когда пользователь будет переключаться на новое поле.
2. Включим проверку корректности введенного электронного адреса. Если введенная текстовая строка не будет содержать символа @, то в диалоговом окне будем выводить предупреждающее сообщение.
3. Сделаем так, чтобы уже при открытии документа поле «Имя» становилось активным. Это удобно, так как пользователь может сразу приступать к заполнению анкеты, не производя никаких дополнительных действий для активизации нужного ему поля ввода.

Текст документа с внесенными в него изменениями приведен ниже:

```
<HTML>
<HEAD>
<TITLE>Анкета с подсказками</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
  function Complete()
  {
    var szStr="";
    szStr="Имя: " + frm.name.value +
          "\nФамилия: " + frm.family.value +
          "\nE-mail: " + frm.email.value;
    alert(szStr);
  }

  function CheckEMail(eml)
```

```
  {
    if (eml != "")
    {
      if (eml.indexOf("@") == -1)
      {
        alert("Внимание!\nЭлектронный адрес указан с
ошибкой.");
      }
    }
  }
// -->
</SCRIPT>
</HEAD>

<BODY>
  <H1>Анкета с подсказками</H1>
  <FORM NAME="frm">
  <TABLE>
  <TR>
    <TD><B>Имя:</B></TD>
    <TD><INPUT NAME="name" TYPE="text" SIZE="15"
         onFocus="comment.innerHTML='Введите ваше имя';">
                                                    </TD>
    <TD rowspan="4" valign="top"><div id="comment">
                  Пожалуйста, заполните анкету</div></TD>
  </TR>
  <TR>
    <TD><B>Фамилия:</B></TD>
    <TD><INPUT NAME="family" TYPE="text" SIZE="15"
     onFocus="comment.innerHTML='Введите вашу фамилию';">
                                                    </TD>
  </TR>
  <TR>
    <TD><B>E-mail:</B></TD>
    <TD><INPUT NAME="email" TYPE="text" SIZE="20"
         onFocus="comment.innerHTML='Введите адрес вашей
электронной почты';"
                 onChange="CheckEMail(this.value);"></TD>
  </TR>
  <TR>
    <TD> </TD>
    <TD><INPUT VALUE="Показать" TYPE="button"
                              onClick="Complete();"></TD>
  </TR>
  </TABLE>
  </FORM>
  <SCRIPT LANGUAGE="JavaScript">
  <!--
    document.frm.name.focus();
```

```
  //-->
  </SCRIPT>
</BODY>
</HTML>
```

Для решения первой задачи (подсказки для каждого текстового поля) содержимое правой ячейки таблицы с помощью тега `<div>` было оформлено в виде отдельного слоя:

```
<TD rowspan="4" valign="top"><div id="comment">Пожалуйста,
                                заполните анкету</div></TD>
```

Этому слою назначен идентификатор `comment`, который позволяет обращаться к нему из сценариев JavaScript. В нашем случае это потребовалось для динамической замены содержимого слоя (через свойство `innerHTML`) для вывода контекстных подсказок при выборе пользователем очередного текстового поля для заполнения. Данная идея реализована через обработку событий `onFocus` каждого текстового поля:

```
<INPUT NAME="name" TYPE="text" SIZE="15"
         onFocus="comment.innerHTML='Введите ваше имя';">
. . . . .
<INPUT NAME="family" TYPE="text" SIZE="15"
     onFocus="comment.innerHTML='Введите вашу фамилию';">
. . . . .
<INPUT NAME="email" TYPE="text" SIZE="20"
onFocus="comment.innerHTML='Введите адрес вашей электронной
почты';" ...>
```

В результате перемещение пользователем курсора по текстовым полям приводит и к замене подсказки в правой части окна анкеты. Заметим, что после решения третьей задачи (автоматический фокус для первого текстового поля), стандартная фраза «Пожалуйста, заполните анкету» пользователю никогда не показывается, так как событие `onFocus` для первого текстового поля возникает автоматически.

Для проверки корректности введенного электронного адреса (вторая задача) была добавлена функция `CheckEMail`:

```
  function CheckEMail(eml)
  {
    if (eml != "")
    {
      if (eml.indexOf("@") == -1)
      {
        alert("Внимание!\nЭлектронный адрес указан с
ошибкой.");
      }
```

```
    }
  }
```

Эта функция проверяет строку, переданную через параметр `eml`. Если эта строка не пустая, но символ @ не содержит, то считается, что электронный адрес введен ошибочно, и в этом случае выводится предупреждающее сообщение.

Вызов функции `CheckEMail` производится при возникновении события `onChange` для текстового поля `email`. В качестве параметра этой функции передается `this.value` (свойство `value` того объекта, который инициализирует вызов функции), то есть введенная строка:

```
<INPUT NAME="email" TYPE="text" SIZE="20" ...
                  ... onChange="CheckEMail(this.value);">
```

Таким образом, проверка электронного адреса производится всякий раз при завершении его ввода (при изменении текста, нажатии клавиши Enter или снятии фокуса с поля ввода электронного адреса). Заметим, что эта проверка выполняется лишь один раз, и если введенный адрес (даже ошибочный) изменению не подвергался, то повторных ошибочных сообщений выдаваться не будет.

Для решения последней задачи (автоматический фокус на поле «Имя») был добавлен следующий фрагмент сценария:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.frm.name.focus();
//-->
</SCRIPT>
```

Это фрагмент добавлен в тело документа HTML сразу после определения формы. В результате, этот сценарий вызывается лишь один раз после создания формы и всех ее элементов и вызывает метод `focus()` для поля `name` формы `frm`. Еще раз заметим, что это не только активизирует поле «Имя», но и вызывает для него событие `onFocus`, что приводит к выводу соответствующей контекстной подсказки.

Поле для ввода пароля password

Поле для ввода пароля по своему устройству в целом аналогично однострочному текстовому полю. Его главные отличия в том, что в поле не отображается набранный текст (обычно все символы заменяются на * или аналогичный символ), и это поле не может иметь обработчиков событий.

Как и многие другие элементы форм, поле password устанавливается с помощью оператора <INPUT>:

```
<INPUT TYPE="password"
  NAME="Имя_поля_password"
  VALUE="Значение_по_умолчанию"
  SIZE="Размер_поля"
  MAXLENGTH="Максимальное_число_символов">
```

Значения параметров поля password полностью аналогичны соответствующим параметрам поля text. Кроме того, объект password имеет и аналогичные объекту text свойства defaultValue, name, value и методы focus(), blur(), select(). Описания этих свойств и методов приведены выше.

Пример 2.4. «Секретная» страница

В качестве иллюстрации работы поля password приведем пример организации парольного доступа к информации, расположенной на странице HTML. Сразу отметим, что по-настоящему надежный парольный доступ можно организовать лишь с использованием серверных приложений, да и то лишь при условии использования защищенных каналов связи. В нашем случае (при использовании сценариев, осуществляющих проверку на стороне клиента) защищенность может быть лишь внешняя, так как вся «секретная» информация (в том числе – пароли) будет доступна через просмотр исходного кода страницы, что, однако, может быть приемлемо во многих случаях.

Исходный код страницы с «секретным» разделом приведен ниже:

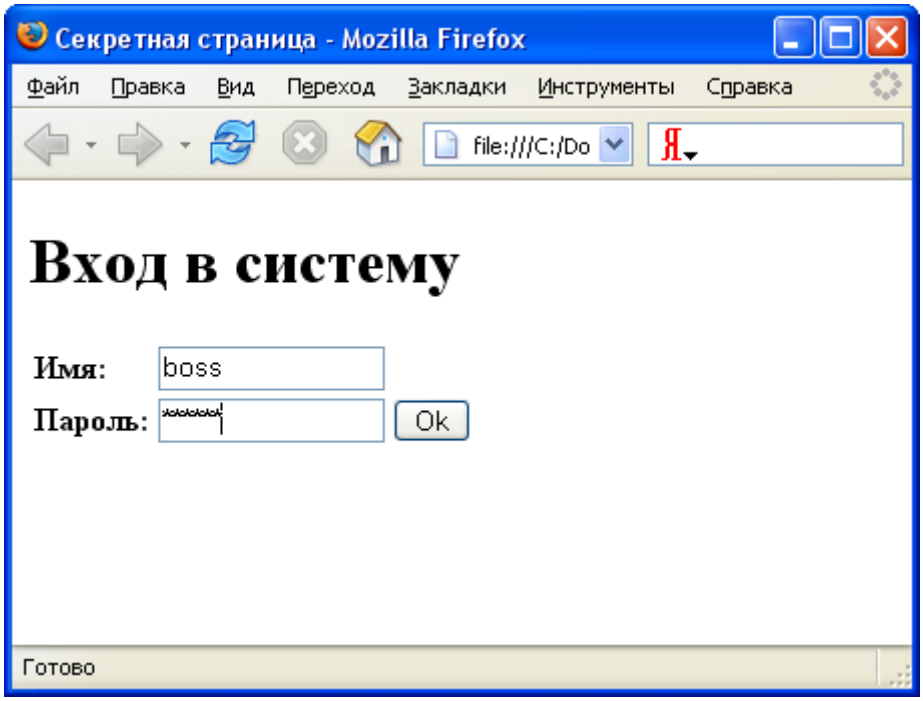
```
<HTML>
<HEAD>
<TITLE>Секретная страница</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function LoginOk()
{
  names = new Array();
  passwds = new Array();
  names[0] = "boss";
  passwds[0] = "mypass";
  names[1] = "agent007";
  passwds[1] = "bond";
  names[2] = "buratino";
  passwds[2] = "polechudes";

  login_is_correct = false;
```

```
  for (i=0; i<names.length; i++)
  { if (names[i].toUpperCase ==
        document.loginform.name.value.toUpperCase
        && passwds[i]==document.loginform.passwd.value)
    { login_is_correct = true; } };

  document.all.login.style.display = "none";
  if (login_is_correct)
  { document.all.secret.style.display = "block"; }
  else
  { document.all.denied.style.display = "block"; }
}
//-->
</SCRIPT>
</HEAD>

<BODY>
<DIV id="login" style="display: block;">
<H1>Вход в систему</H1>
<FORM name="loginform">
<TABLE>
<TR>
    <TD><B>Имя:</B></TD>
    <TD><INPUT TYPE="text" NAME="name" SIZE="15"></TD>
    <TD> </TD>
</TR>
<TR>
    <TD><B>Пароль:</B></TD>
    <TD><INPUT TYPE="password" NAME="passwd"
                                        SIZE="15"></TD>
    <TD><INPUT TYPE="button" VALUE="Ok"
                              onClick="LoginOk();"></TD>
</TR>
</TABLE>
</FORM>
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.loginform.name.focus();
//-->
</SCRIPT>
</DIV>

<DIV id="denied" style="display: none;">
<H1>Доступ закрыт</H1>
<P>Имя или пароль указаны неправильно
</DIV>

<DIV id="secret" style="display: none;">
<H1>Секретная страница</H1>
```

```
<P>Новости центра управления разведками
<P><B>.....................</B><BR>
<B>.....................</B>
</DIV>

</BODY>
</HTML>
```

При открытии этого документа в окне браузера отображается форма, где требуется указать имя и пароль пользователя (рисунок 2.5)

Рисунок 2.5. Проверка имени и пароля пользователя



Если имя и пароль указаны корректно, то осуществляется переход к «секретному» разделу (рисунок 2.6), а в противном случае, выводится сообщение о том, что имя или пароль указаны неправильно (рисунок 2.7)

Рисунок 2.6. «Секретный» раздел документа HTML.

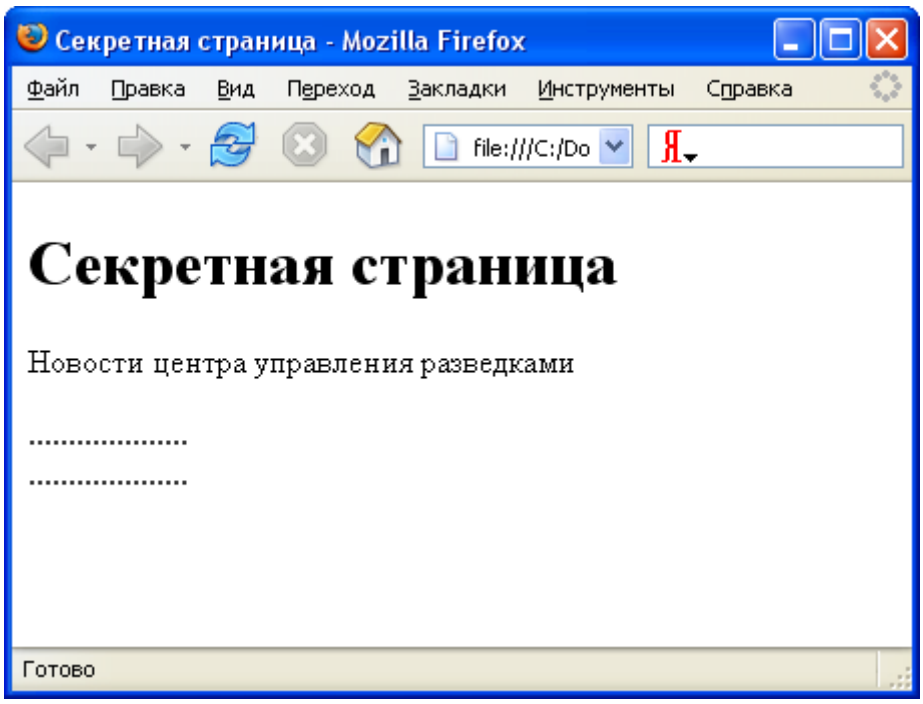
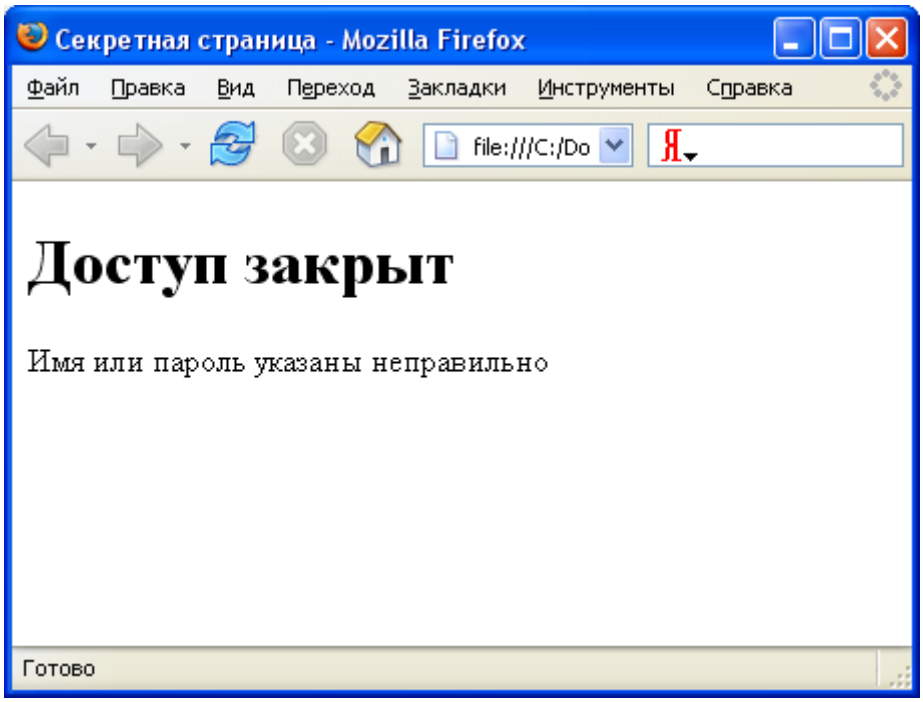


Рисунок 2.7. Сообщение об ошибке в имени или пароле



Рассмотрим структуру документа более подробно. В теле документа представлено три слоя:

```
<DIV id="login" style="display: block;">
<H1>Вход в систему</H1>
... ... ...
</DIV>
```

```
<DIV id="denied" style="display: none;">
<H1>Доступ закрыт</H1>
... ... ...
</DIV>

<DIV id="secret" style="display: none;">
<H1>Секретная страница</H1>
... ... ...
</DIV>
```

Для второго и третьего слоя указан параметр `style` со значением `"display: none;"`. Это означает, что слои 2 и 3 изначально скрыты и при открытии страницы в окне браузера не отображаются.

Первый слой – видимый, и он содержит форму с полями «Имя», «Пароль» и кнопкой «Ok». Сценарий, расположенный сразу после формы, делает активным поле «Имя».

Проверка корректности введенного имени и пароля производится с помощью функции `LoginOk()`, которая назначена как обработчик события для кнопки «Ok».

```
function LoginOk()
{
  names = new Array();
  passwds = new Array();
  names[0] = "boss";
  passwds[0] = "mypass";
  names[1] = "agent007";
  passwds[1] = "bond";
  names[2] = "buratino";
  passwds[2] = "polechudes";

  login_is_correct = false;
  for (i=0; i<names.length; i++)
  { if (names[i].toUpperCase ==
        document.loginform.name.value.toUpperCase
        && passwds[i]==document.loginform.passwd.value)
    { login_is_correct = true; } };

  document.all.login.style.display = "none";
  if (login_is_correct)
  { document.all.secret.style.display = "block"; }
  else
  { document.all.denied.style.display = "block"; }
}
```

Эта функция в массивах `names` и `passwds` содержит списки пользователей и соответствующих им паролей:

```
names[0] = "boss";
passwds[0] = "mypass";
names[1] = "agent007";
passwds[1] = "bond";
names[2] = "buratino";
passwds[2] = "polechudes";
```

В данном примере хранятся сведения о трех пользователях, и этот список можно произвольным образом модифицировать и расширять.

Далее следует проверка имени и пароля, указанных пользователем:

```
login_is_correct = false;
for (i=0; i<names.length; i++)
{ if (names[i].toUpperCase ==
      document.loginform.name.value.toUpperCase
      && passwds[i]==document.loginform.passwd.value)
  { login_is_correct = true; } };
```

С помощью цикла `for` перебираются все элементы массивов `names` и `passwds` и сравниваются со значениями, введенными пользователем. Если совпадение было обнаружено, то переменной `login_is_correct` присваивается значение `true`.

Заметим, что сравнение имен производится в верхнем регистре (используется метод `toUpperCase`). Это позволяет сделать имена регистронезависимыми (имена вида **peter**, **Peter**, **PETER** понимаются как одинаковые). При проверке пароля подобное преобразование не производится и регистр вводимых букв значение иметь будет.

После проверки имени и пароля следует завершающий блок:

```
document.all.login.style.display = "none";
if (login_is_correct)
{ document.all.secret.style.display = "block"; }
else
{ document.all.denied.style.display = "block"; }
```

Здесь скрывается слой с формой ввода имени и пароля и на основе данных проверки (используется переменная `login_is_correct`) отображается слой с «секретной» информацией или слой с указанием ошибки во введенных данных. Таким образом, пользователь при правильном указании имени и пароля получает доступ к «секретному» разделу.

Многострочное текстовое поле textarea

Многострочное текстовое поле используется в тех случаях, когда редактируемый текст должен занимать несколько строк. Для его создания используется специальный парный тег <TEXTAREA>:

```
<TEXTAREA
  NAME="Имя_поля_textarea"
  ROWS="Количество_строк"
  COLS="Количество_столбцов"
  WRAP="Режим_свертки_текста"
  onBlur="Обработчик_события"
  onChange="Обработчик_события"
  onFocus="Обработчик_события"
  onSelect="Обработчик_события">

    Отображаемый текст

</TEXTAREA>
```

Как обычно, с помощью параметра NAME указывается имя поля. Параметры ROWS и COLS определяют видимый размер многострочного поля редактирования, задавая, соответственно, количество строк и столбцов (количество символов, которые могут поместиться в одной строке).

Параметр WRAP задает способ свертки текста (переноса строк) и может иметь одно из трех следующих значений:

Значение	***Описание***
off	Свертка выключена, строки отображаются так, как вводятся
virtual	Строки сворачиваются только при отображении в окне редактирования, но передаются расширению сервера Web и сценарию JavaScript точно в таком виде, в котором вводятся
physical	При свертке в передаваемый текст записываются символы новой строки

Значение по умолчанию (отображаемый текст) здесь задается в явном виде между тегами <TEXTAREA> ... </TEXTAREA>. Несмотря на это, из сценариев JavaScript доступ к введенному тексту производится так же, как и в случае работы с однострочным тестовым полем – через свойства value и defaultValue.

Для объекта textarea определен и такой же набор свойств и методов, что и для объекта text. Это уже знакомые методы focus(), blur() и select(), а также события onFocus, onBlur, onChange и onSelect.

Работа с многострочными текстовыми полями в целом аналогична работе с полями типа text, и фрагмент сценария, осуществляющего

обработку данных из полей textarea, будет представлен в одном из следующих примеров.

Переключатель checkbox

Переключатель checkbox («флажок», «галочка») представляет собой элемент управления, позволяющий устанавливать пользователю одно из двух значений («включено», «выключено»). Часто используется сразу несколько переключателей, что позволяет выбирать какие-либо независимые друг от друга параметры или возможности.

В форме переключатель checkbox создается с помощью оператора <INPUT> с параметром TYPE со значением "checkbox":

```
<INPUT TYPE="checkbox"
  NAME="Имя_переключателя_checkbox"
  VALUE="Значение"
  CHECKED
  onClick="Обработчик_события">
```

Параметр NAME задает имя переключателя. Это имя можно использовать для определения состояния этого переключателя в сценарии JavaScript.

С помощью параметра VALUE можно определить строку, которая передается расширению сервера при посылке заполненной формы, если переключатель находится во включенном состоянии. Если этот параметр не указан, то по умолчанию посылается строка "on". Сценарий JavaScript также может получить значение параметра VALUE, однако, как правило, это не используется, так как это значение не отражает текущее состояние переключателя.

Отметим также, что параметр VALUE никак не влияет и на внешний вид переключателя на странице HTML. Если надо задать текст, поясняющий назначение переключателя, то он задается обычными средствами HTML, как и весь остальной текст, расположенный на странице.

Необязательный параметр CHECKED указывается в том случае, если при начальном отображении формы переключатель должен отображаться во включенном состоянии.

Событие onClick позволяет задать сценарий JavaScript, получающий управление после того как пользователь изменит состояние переключателя.

Свойства и методы объекта checkbox

Объект checkbox имеет несколько свойств, отражающих значения соответствующих параметров оператора <INPUT>:

Свойство	*Описание*
name	Значение параметра NAME
value	Значение параметра VALUE
checked	Свойство типа Boolean, отражающее состояние переключателя. Если переключатель включен, свойство имеет значение true, в противном случае – false. С помощью этого свойства сценарий может и изменять состояние переключателя
defaultChecked	Свойство типа Boolean, отражающее наличие параметра CHECKED. Если параметр CHECKED присутствует в определении переключателя, свойство имеет значение true, в противном случае – false

Для объекта checkbox определен один метод click(), не имеющий параметров. При вызове этого метода переключатель устанавливается во включенное состояние.

Пример 2.5. Анкета с выбором всех правильных вариантов

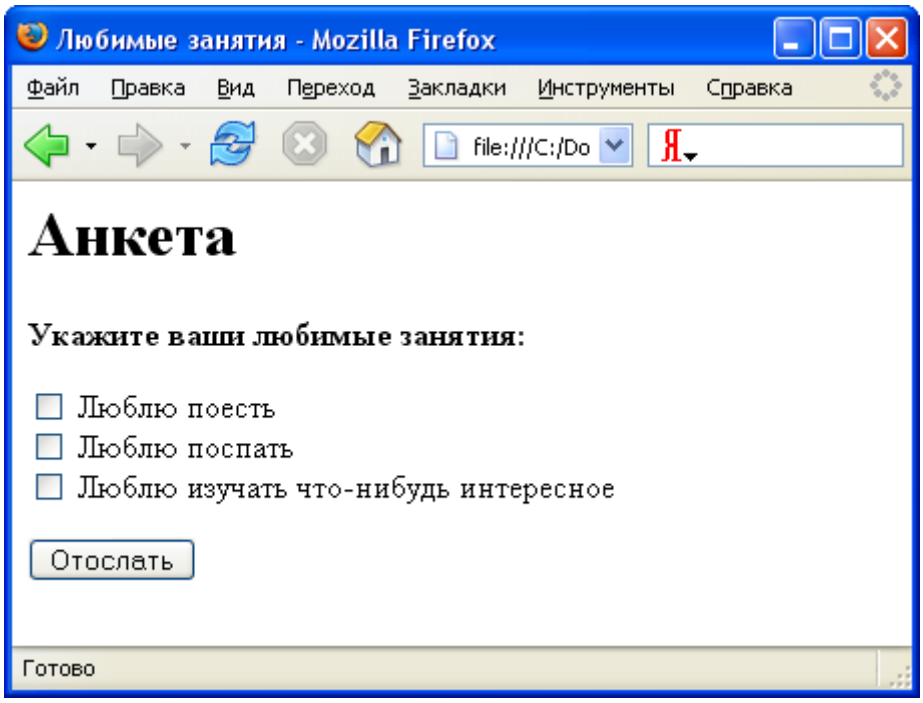
Рассмотрим использование переключателей checkbox на примере простой анкеты, в которой предлагается выбрать все подходящие ответы из трех предложенных.

```
<HTML>
<HEAD>
<TITLE>Любимые занятия</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function Complete()
{
  var szTxtHead ="Ваши любимые занятия:\n";
  var szTxtAnswer ="";

  if (document.anketa.eda.checked)
    {szTxtAnswer += "люблю поесть\n"};
  if (document.anketa.son.checked)
    {szTxtAnswer += "люблю поспать\n"};
  if (document.anketa.izuchat.checked)
    {szTxtAnswer += "люблю изучать что-нибудь интересное"};

  if (szTxtAnswer == "")
    {szTxtAnswer = "нет любимых занятий"};

  alert(szTxtHead + szTxtAnswer);
}
```

```
//-->
</SCRIPT>
</HEAD>

<BODY>
<H1>Анкета</H1>
<P><B>Укажите ваши любимые занятия:</B>
<FORM NAME="anketa">
<INPUT TYPE="checkbox" NAME="eda">
Люблю поесть<BR>
<INPUT TYPE="checkbox" NAME="son">
Люблю поспать<BR>
<INPUT TYPE="checkbox" NAME="izuchat">
Люблю изучать что-нибудь интересное
<P><INPUT TYPE="button" VALUE="Отослать"
                        onClick="Complete();">
</FORM>
</BODY>
</HTML>
```

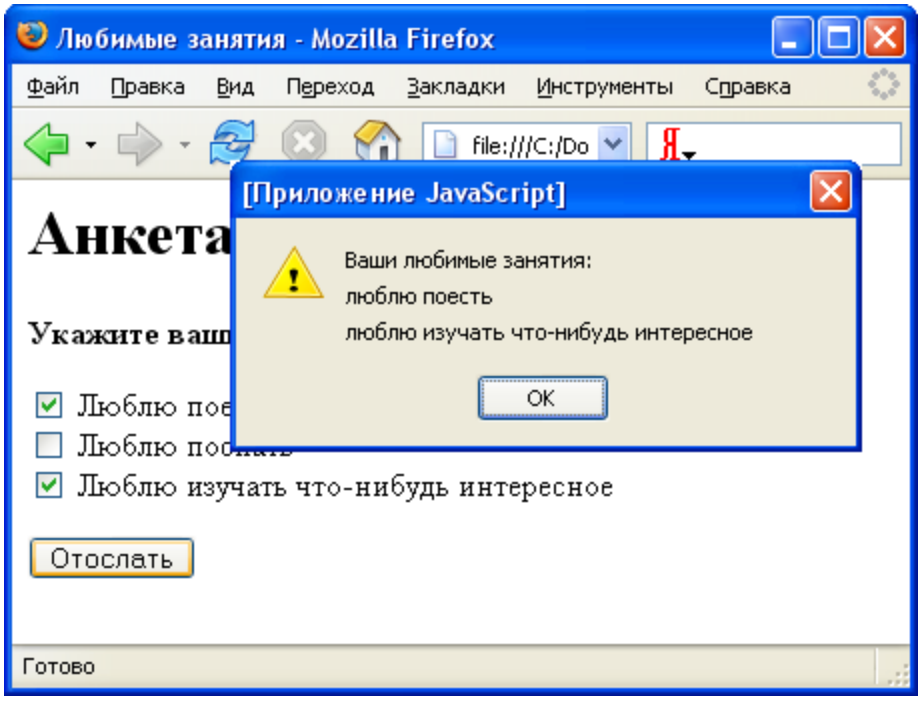
Внешний вид представленной HTML-страницы показан на рисунке 2.8. Обратите внимание, что при описании переключателей `checkbox` нигде не был указан параметр `CHECKED` и вследствие этого все переключатели находятся в состоянии «выключено».

Рисунок 2.8. Документ HTML с переключателями checkbox



Можно поставить отметку в любые переключатели на странице и нажать кнопку «Отослать», в результате чего будет выведено диалоговое окно с указанием отмеченных пользователем строк (рисунок 2.9).

Рисунок 2.9. Диалоговое окно с данными анкеты



Обработка данных анкеты производится функцией `Complete()`, которая назначена обработчиком события `onClick` для кнопки «Отослать». Функция, проверяя свойство `checked` для каждого из объектов `checkbox`, определяет их текущее состояние. Если очередной переключатель находится в состоянии «включено» (свойство `checked` имеет значение «истина»), то к текстовой переменной `szTxtAnswer` добавляется соответствующее сообщение:

```
if (document.anketa.eda.checked)
  {szTxtAnswer += "люблю поесть\n"};
if (document.anketa.son.checked)
  {szTxtAnswer += "люблю поспать\n"};
if (document.anketa.izuchat.checked)
  {szTxtAnswer += "люблю изучать что-нибудь интересное"};
```

Если переменная `szTxtAnswer` после этих проверок оказалась пустой (пользователь не отметил ни один из переключателей), то ей присваивается фраза «нет любимых занятий».

В завершение функции вызывается метод `alert` для вывода диалогового окна с обработанными данными анкеты.

Переключатель radio

Переключатели типа `radio` применяются в тех случаях, когда нужно организовать выбор только одной из нескольких возможностей. Исходя из этого, в форме обычно располагается несколько таких переключателей, которые образуют группу. В каждой группе не более одного переключателя `radio` может иметь состояние «включено».

Определение переключателя `radio` выглядит следующим образом:

```
<INPUT TYPE="radio"
  NAME="Имя_переключателя_radio"
  VALUE="Значение"
  CHECKED
  onClick="Обработчик_события">
```

Назначение параметров `NAME`, `VALUE` и `CHECKED` переключателя `radio` такое же, как и у переключателя `checkbox`. Отличие заключается в том, что все переключатели `radio`, принадлежащие к одной группе, должны иметь **одинаковые имена**, определенные параметром `NAME` (все переключатели `checkbox` должны называться по-разному). Но, для того чтобы расширение сервера Web или сценарий JavaScript, обрабатывающий форму, могли узнать, какой же из переключателей `radio` группы находится во включенном состоянии, все такие переключатели должны иметь **различные значения** `VALUE` (значения параметров `VALUE` переключателей `checkbox` могут быть одинаковыми). Кроме того, только один из переключателей `radio` может быть определен с параметром `CHECKED`.

Свойства и методы объекта radio

В таблице представлены свойства объекта `radio`:

Свойство	*Описание*
`name`	Значение параметра `NAME`
`value`	Значение параметра `VALUE`. Доступно только при обращении к конкретному переключателю `radio`, а не ко всей группе, определенной значением параметра `NAME`.
`length`	Количество переключателей типа `radio`, определенных в группе с именем, заданным параметром `NAME`
`checked`	Свойство типа `Boolean`, отражающее состояние переключателя. Если переключатель включен, свойство имеет значение `true`, в противном случае –

	false. С помощью этого свойства сценарий может и изменять состояние переключателя
defaultChecked	Свойство типа Boolean, отражающее наличие параметра CHECKED. Если параметр CHECKED присутствует в определении переключателя, свойство имеет значение true, в противном случае – false

Для объекта radio определен метод click(), не имеющий параметров. При вызове этого метода переключатель выбирается для работы.

Следует обратить внимание на способ обращения к конкретным переключателям radio. Нестандартность ситуации заключается в том, что параметр NAME определяет не конкретный переключатель (как это было с другими элементами форм), а целую группу переключателей и свойство name определяет их массив. Доступ к переключателям осуществляется как к элементам массива, что позволяет обращаться к свойствам конкретных переключателей (например, к value или checked).

Пример 2.6. Анкета с выбором единственного правильного варианта

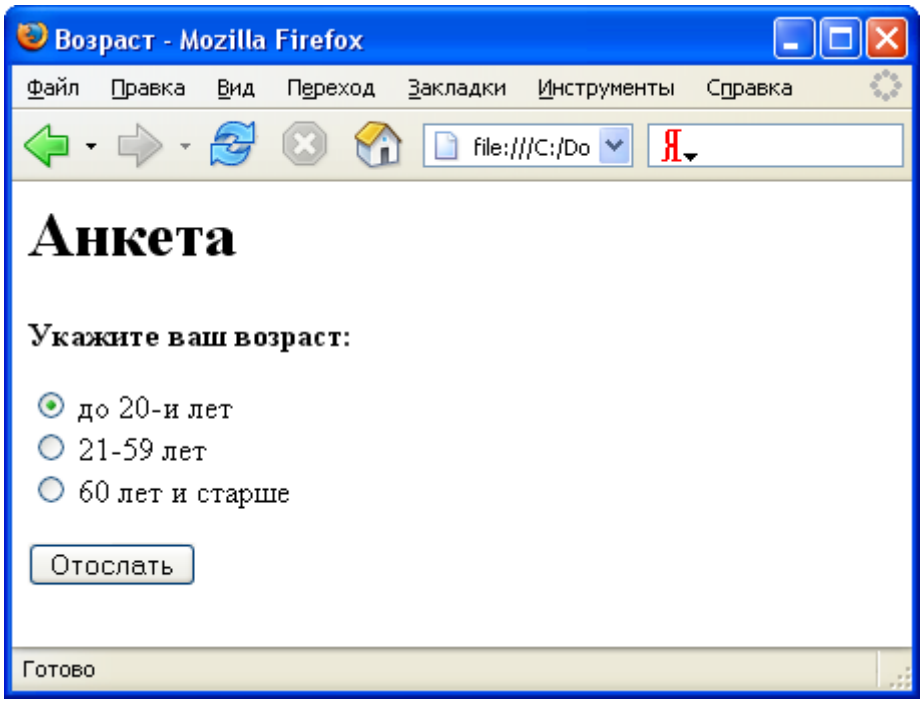
Рассмотрим простейший пример анкеты, в которой необходимо выбрать единственный правильный вариант ответа. Эта анкета построена с использованием переключателя radio.

```
<HTML>
<HEAD>
<TITLE>Возраст</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function Complete()
{
  var szHead ="Ваш возраст: ";
  var szAnswer = "";
  for (i=0; i<document.anketa.age.length; i++)
  {
    if (document.anketa.age[i].checked)
      {
        szAnswer=document.anketa.age[i].value;
      };
  };
  alert(szHead + szAnswer);
}
//-->
</SCRIPT>
</HEAD>
```

```
<BODY>
<H1>Анкета</H1>
<P><B>Укажите ваш возраст:</B>
<FORM NAME="anketa">
<INPUT TYPE="radio" NAME="age" VALUE="до 20-и лет" CHECKED>
до 20-и лет<BR>
<INPUT TYPE="radio" NAME="age" VALUE="21-59 лет">
21-59 лет<BR>
<INPUT TYPE="radio" NAME="age" VALUE="60 лет и старше">
60 лет и старше<BR>
<P><INPUT TYPE="button" VALUE="Отослать"
                                    onClick="Complete();">
</FORM>
</BODY>
</HTML>
```

Приведенный выше документ HTML имеет в своем составе форму с группой из трех переключателей `radio`:

Рисунок 2.10. Переключатели radio на странице HTML



Выбор любого из неотмеченных переключателей этой формы автоматически приводит к снятию старой отметки. В результате этого, отмеченным переключателем может только один. Обратите внимание, что для определения согласованно работающей **группы** переключателей `radio` в их описании указано одно и то же имя – `age`:

```
<INPUT TYPE="radio" NAME="age" VALUE="до 20-и лет" CHECKED>
... ... ...
```

```
<INPUT TYPE="radio" NAME="age" VALUE="21-59 лет">
... ... ...
<INPUT TYPE="radio" NAME="age" VALUE="60 лет и старше">
```

Если нажать кнопку «Отослать», то будет выведено диалоговое окно с указанием отмеченного варианта ответа. Как и в прошлом примере, эта обработка анкетных данных производится с помощью функции `Complete()`. Для проверки состояния всех переключателей группы `age` используется цикл, который позволяет обратиться к каждому объекту `radio` как к элементу массива `document.anketa.age`:

```
for (i=0; i<document.anketa.age.length; i++)
{
  if (document.anketa.age[i].checked)
    {
      szAnswer=document.anketa.age[i].value;
    };
};
```

Таким образом, в цикле проверяется состояние всех переключателей `radio`, представленных на странице, и переменной `szAnswer` присваивается значение свойства `value` того элемента, который имеет состояние «включено». Далее, строка `szAnswer` выводится в диалоговом окне, вызываемом методом `alert`.

Список выбора select

Список выбора представляет собой элемент управления, состоящий из нескольких текстовых строк, предлагаемых пользователю в качестве альтернативы. Список выбора может выглядеть как однострочное поле с прокруткой и выпадающим меню, а может предлагать сразу несколько строк, размещенных в окне с прокруткой. В зависимости от настроек, выбор пользователя может быть как единственным, так и множественным (дополнительные строки выбираются мышью при нажатой клавише **Ctrl** или **Shift** на клавиатуре).

В любом случае, размещение списка выбора производится с помощью парного тега `<SELECT>`, формат которого приведен ниже:

```
<SELECT
  NAME="Имя_списка_select"
  SIZE="Количество_строк"
  MULTIPLE
  onBlur="Обработчик_события"
  onChange="Обработчик_события"
  onFocus="Обработчик_события">
```

```
    <OPTION VALUE="Значение" SELECTED>Строка 1
    <OPTION VALUE="Значение">Строка 2
    ...
</SELECT>
```

Все параметры оператора <SELECT> необязательные, однако для того чтобы сценарий JavaScript мог работать со списком, необходимо указать по крайней мере параметр NAME, определяющий имя списка.

Параметр SIZE задает количество строк, видимых на экране. Если SIZE имеет значение 1 (значение по умолчанию), то список выбора снабжается выпадающим меню. Если SIZE больше одного, то строки для выбора размещаются в окне с прокруткой. Заметим, что ширина списка определяется по содержимому и рассчитывается так, чтобы в списке смогла разместиться самая длинная строка.

Если указан параметр MULTIPLE, то список выбора позволяет производить множественный выбор. В противном случае (значение по умолчанию), список выбора работает по аналогии с группой переключателей radio, позволяя отмечать только один вариант из числа предложенных.

Элементы списка определяются с помощью вложенного в парный тег <SELECT> операторов <OPTION>.

Оператор <OPTION> может иметь два параметра – VALUE и SELECTED. Параметр VALUE на внешний вид списка не влияет и определяет значение, которое передается расширению сервера Web или обрабатывается сценарием JavaScript. С помощью параметра SELECTED отмечаются строки списка, выделенные по умолчанию при начальном отображении формы (если в параметрах <SELECT> не указано MULTIPLE, то SELECTED можно использовать не более одного раза). Текст, отображаемый в строках списка, размещается сразу после оператора <OPTION>.

Свойства объекта select

Ниже перечислены свойства объекта select, доступные сценарию JavaScript:

Свойство	*Описание*
name	Значение параметра NAME
length	Количество элементов (строк) в списке
selectedIndex	Номер выбранного элемента или первого элемента среди нескольких выбранных (если указан параметр MULTIPLE и пользователь выбрал в списке несколько элементов)
options	Массив объектов, соответствующих элементам списка, заданным при помощи оператора <OPTION>

Как показано в таблице, одним из свойств списка select является массив options. В этом массиве хранятся элементы списка, определенные оператором <OPTION>. Каждый элемент такого массива есть объект класса Option со следующим набором свойств:

Свойство	*Описание*
name	Значение параметра NAME
value	Значение параметра VALUE
text	Текст, указанный после оператора <OPTION>
index	Порядковый номер (индекс) элемента списка
selected	Свойство типа Boolean, отражающее состояние элемента списка. Если элемент выбран, свойство имеет значение true, в противном случае – false. Свойство доступно и для записи, то есть с его помощью сценарий может выбрать элемент
defaultSelected	Свойство типа Boolean, отражающее наличие параметра SELECTED. Если параметр SELECTED присутствует в определении элемента списка, свойство имеет значение true, в противном случае – false

Массив options доступен и для изменения. Добавление к массиву новых элементов приводит к расширению списка.

Методы объекта select

Для объекта select определено два метода, не имеющих параметров, - focus() и blur(). Первый из этих методов позволяет передать списку фокус ввода, а второй - отобрать этот фокус у списка.

Обработчики событий, связанные с объектом select

Как видно из формата оператора <SELECT>, для списка можно определить три обработчика события: onFocus, onBlur и onChange. События onFocus и onBlur возникают, когда список получает и теряет фокус ввода, соответственно. Событие onChange создается, когда пользователь изменяет состояние списка, то есть выбирает в нем другой элемент.

Пример 2.7. Список быстрой навигации

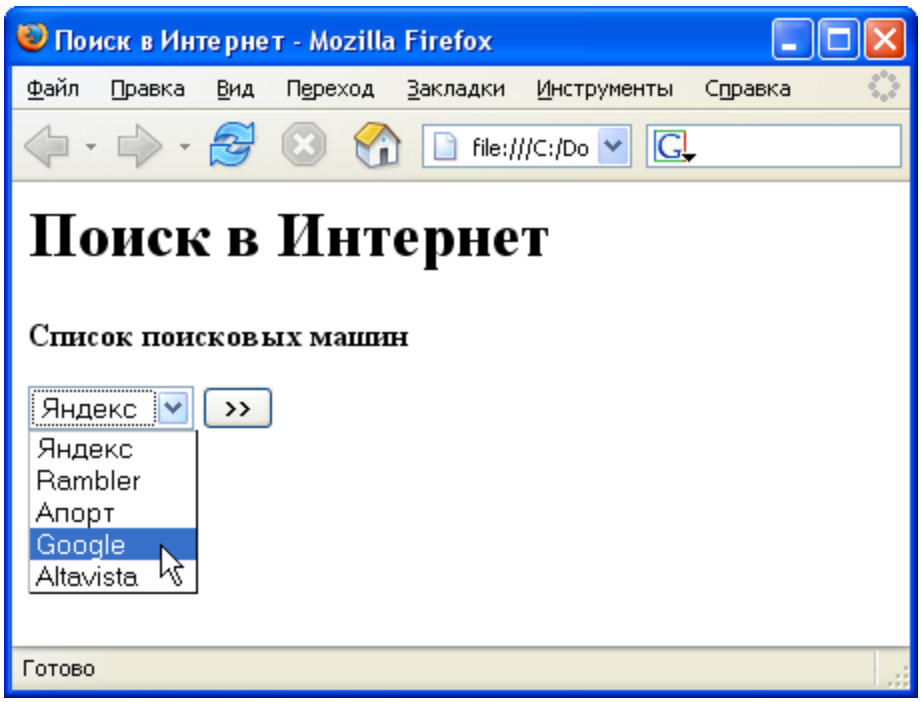
В качестве примера взаимодействия сценария JavaScript со списком select, допускающим выбор только одного варианта, рассмотрим реализацию популярных в настоящее время списков быстрой навигации,

позволяющих осуществлять переход к другим разделам сервера (или к другим серверам).

```
<HTML>
<HEAD>
<TITLE>Поиск в Интернет</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function Complete()
{
  var nSelectedIndex =
     document.frm.serverlist.selectedIndex;
  var szURL =
     document.frm.serverlist.options[nSelectedIndex].value;
  window.location = szURL;
}
//-->
</SCRIPT>
</HEAD>

<BODY>
<H1>Поиск в Интернет</H1>
<P><B>Список поисковых машин</B>
<FORM NAME="frm">
  <SELECT NAME="serverlist">
    <OPTION VALUE="http://www.yandex.ru" SELECTED> Яндекс
    <OPTION VALUE="http://www.rambler.ru"> Rambler
    <OPTION VALUE="http://www.aport.ru"> Апорт
    <OPTION VALUE="http://www.google.com"> Google
    <OPTION VALUE="http://www.altavista.com"> Altavista
  </SELECT>
  <INPUT TYPE="button" VALUE=">>" onClick="Complete();">
</FORM>
</BODY>
</HTML>
```

На рисунке 2.11 показан внешний вид созданного нами списка быстрой навигации. Список представлен в развернутом виде, после нажатия пользователем клавиши выпадающего меню. Пользователю предлагается выбрать интересующую его поисковую машину и нажать клавишу с двойной стрелкой для перехода к соответствующему серверу.

Рисунок 2.11. Список быстрой навигации



Рассмотрим детали реализации представленного списка. Список создан с помощью оператора `<SELECT>`, расположенного в форме документа HTML:

```
<SELECT NAME="serverlist">
  <OPTION VALUE="http://www.yandex.ru" SELECTED> Яндекс
  <OPTION VALUE="http://www.rambler.ru"> Rambler
  <OPTION VALUE="http://www.aport.ru"> Апорт
  <OPTION VALUE="http://www.google.com"> Google
  <OPTION VALUE="http://www.altavista.com"> Altavista
</SELECT>
```

Обратите внимание, что в определении списка не указаны параметры `SIZE` и `MULTIPLE`. Это означает, что список должен быть однострочным (с выпадающим меню) и позволяющим выбирать только одно значение из предлагаемых.

Элементы списка оформлены с помощью операторов `<OPTION>`. Для каждого элемента задана отображаемая на экране строка (название поисковой машины) и ее адрес. Адрес для каждой машины записан как значение параметра `VALUE`. Адреса никак не отображаются в окне документа HTML, но впоследствии используются сценарием JavaScript для осуществления перехода к нужным серверам.

Форма списка быстрой навигации содержит также кнопку, при нажатии на которую вызывается функция `Complete()`, которая определяет указанный пользователем поисковый сервер и осуществляет переход к нему:

```
function Complete()
```

```
{
  var nSelectedIndex =
     document.frm.serverlist.selectedIndex;
  var szURL =
     document.frm.serverlist.options[nSelectedIndex].value;
  window.location = szURL;
}
```

Определение выбранного пользователем элемента списка производится в два этапа. На первом этапе определяется номер выбранного элемента:

```
  var nSelectedIndex =
     document.frm.serverlist.selectedIndex;
```

Далее, происходит обращение к соответствующему элементу массива `options` (элементу списка) и определяется значение параметра `value` (адрес поисковой машины):

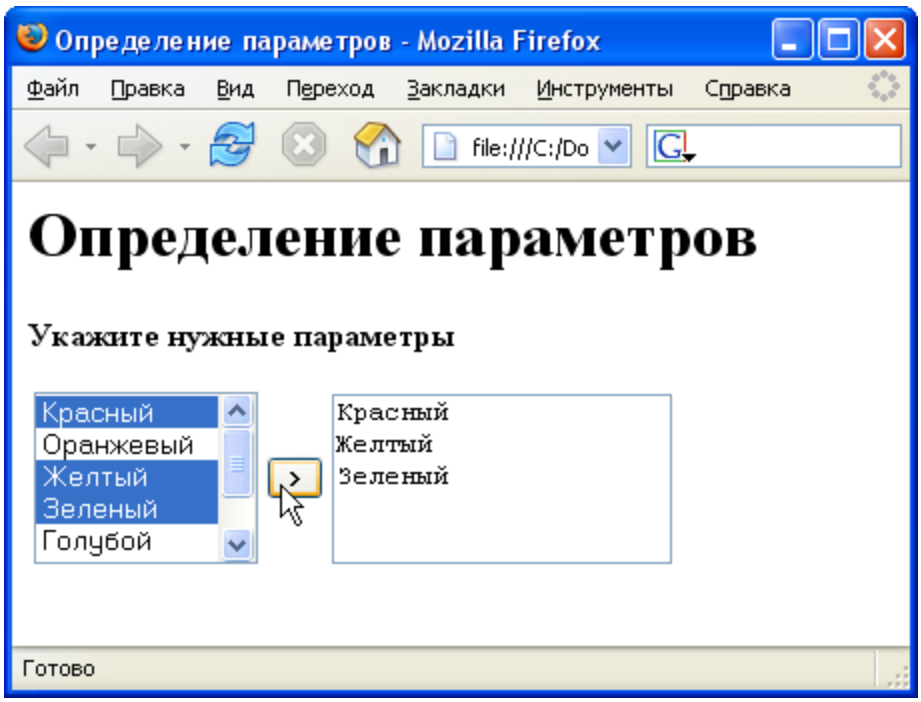
```
  var szURL =
     document.frm.serverlist.options[nSelectedIndex].value;
```

Полученная таким образом текстовая строка (адрес поисковой машины) присваивается свойству `location` (адрес текущего документа) объекта `window` (окно браузера), что осуществляет переход к нужной странице.

Пример 2.8. Список для определения набора параметров

Рассмотрим еще один пример работы со списком, в котором используется возможность осуществления множественного выбора. Предположим, что есть набор некоторых параметров (в нашем случае – цвета), оформленных в виде многострочного списка. Пользователю предлагается указать все необходимые из них и нажать кнопку со стрелкой для «переноса» выбранных строк в многострочное текстовое поле, расположенное на этой же странице (рисунок 2.12).

Рисунок 2.12. Определение набора параметров



Исходный текст данной Web-страницы представлен ниже:

```
<HTML>
<HEAD>
<TITLE>Определение параметров</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function Complete()
{
  var szSelectedParam = "";
  for (i=0; i<document.frm.paramlist.length; i++)
  {
    if (document.frm.paramlist.options[i].selected)
     {
       szSelectedParam +=
            document.frm.paramlist.options[i].value + "\n";
     }
  }
  document.frm.paramselected.value = szSelectedParam;
}
//-->
</SCRIPT>
</HEAD>

<BODY>
<H1>Определение параметров</H1>
<P><B>Укажите нужные параметры</B>
<FORM NAME="frm">
<TABLE>
```

```
<TR>
  <TD>
    <SELECT NAME="paramlist" SIZE="5" MULTIPLE>
      <OPTION VALUE="Красный"> Красный
      <OPTION VALUE="Оранжевый"> Оранжевый
      <OPTION VALUE="Желтый"> Желтый
      <OPTION VALUE="Зеленый"> Зеленый
      <OPTION VALUE="Голубой"> Голубой
      <OPTION VALUE="Синий"> Синий
      <OPTION VALUE="Фиолетовый"> Фиолетовый
    </SELECT>
  </TD>
  <TD>
    <INPUT TYPE="button" VALUE=">" onClick="Complete();">
  </TD>
  <TD>
    <TEXTAREA NAME="paramselected"
                ROWS="5" COLS="18"></TEXTAREA>
  </TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Форма в теле документа содержит список `select`, кнопку `button` и многострочное текстовое поле `textarea`. Обратите внимание, что в определении списка указаны параметры `SIZE` со значением `"5"` и `MULTIPLE`. Первый из этих параметров определяет внешний вид списка (многострочный, 5 строк), а второй параметр дает возможность множественного выбора. Пяти строк недостаточно для отображения семи элементов списка, поэтому у списка активен вертикальный скроллинг (рисунок 2.12), который позволяет осуществить прокрутку и получить доступ к изначально скрытым элементам.

Многострочное текстовое поле установлено с помощью парного тега `<TEXTAREA>`. В параметрах этого тега указано имя текстового поля (используется сценарием JavaScript), а также его размеры. Закрывающий тег `</TEXTAREA>` следует сразу за открывающим (между ними нет даже пробелов), то есть на странице текстовое поле изначально пустое.

Обработка указанных пользователем данных производится функцией `Complete()`. Ядром этой функции является цикл, в котором «просматривается» массив `options` списка `paramlist` (перебираются все элементы списка). Если свойство `selected` очередного элемента истинно (элемент был отмечен пользователем), то значение его параметра `value` добавляется к текстовой строке `szSelectedParam`:

```
for (i=0; i<document.frm.paramlist.length; i++)
{
  if (document.frm.paramlist.options[i].selected)
   {
    szSelectedParam +=
         document.frm.paramlist.options[i].value + "\n";
   }
}
```

Сформированная таким образом текстовая строка присваивается параметру `value` многострочного текстового поля (объект `paramselected` формы `frm`):

```
document.frm.paramselected.value = szSelectedParam;
```

Данное присвоение обеспечивает отображение в текстовом поле выбранных пользователем параметров из списка.

Раздел 3. Дополнительные поля и органы управления

Итак, нами рассмотрены основные поля и органы управления, которые могут входить в состав форм документов HTML и использоваться при локальной обработке данных сценариями JavaScript. Вместе с тем, этот список не исчерпывающий. Существует еще несколько элементов, которые могут содержаться в формах. Мы обошли их вниманием по той причине, что эти элементы предназначены в основном для работы с приложениями, выполняющимися на стороне сервера. К числу таких элементов можно отнести:

- Графическая кнопка `image`;
- Поле выбора файлов `file`;
- Скрытое поле `hidden`.

Исключением из этого списка может быть кнопка, устанавливаемая оператором `<BUTTON>`, позволяющая использовать графическое изображение на своей поверхности.

Коротко остановимся на особенностях данных элементов и их использовании в документах Web.

Графическая кнопка image

Графическая кнопка image является аналогом кнопки submit, которая инициирует пересылку данных, введенных пользователем, серверу Web. Особенность этой кнопки в том, что она позволяет использовать любое графическое изображение для своего отображения в окне браузера.

Общий вид оператора кнопки image с основными параметрами приведен ниже:

```
<INPUT TYPE="image"
  NAME="Имя_кнопки"
  SRC="Файл_с_графическим_изображением">
```

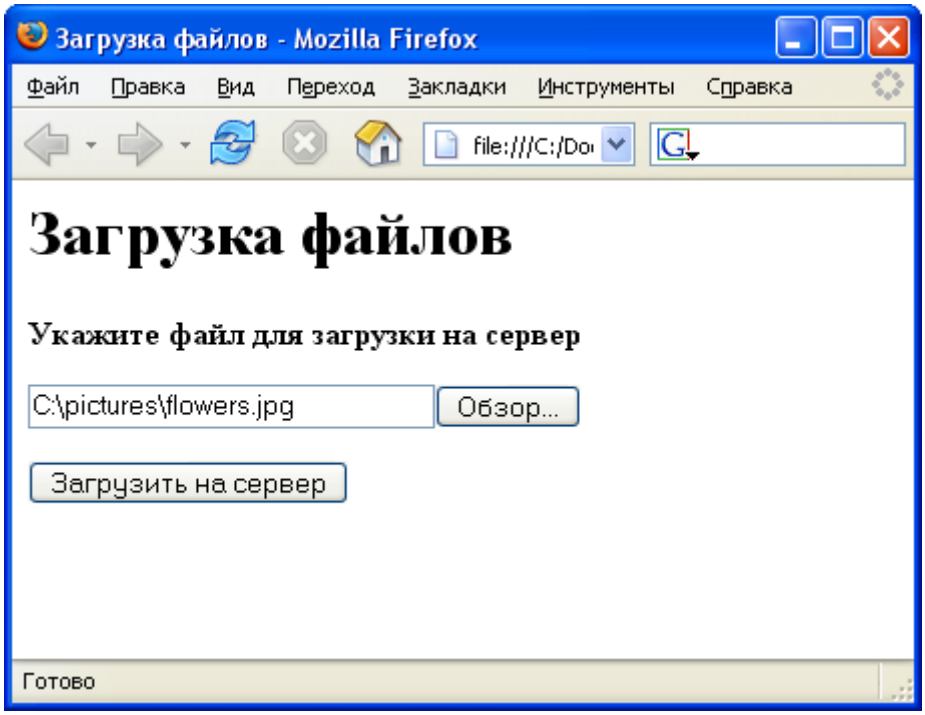
Параметры TYPE и NAME стандартны для элементов INPUT. В параметре SRC указывается путь к файлу с графическим изображением, которое надо использовать в качестве внешнего представления кнопки в окне браузера. Кроме этих параметров, в описании графической кнопки можно использовать многие параметры тега <IMG>, управляющие внешним видом изображения. Это такие параметры, как WIDTH, HEIGHT, BORDER, ALT и другие.

Обратите внимание, что в описании отсутствует параметр VALUE. Дело в том, что на внешний вид кнопки он влияния не оказывает, а расширению сервера Web передается конструкция вида Имя_кнопки.x=XX&Имя_кнопки.y=YY, где XX и YY – это координаты указателя мыши в момент щелчка относительно левого верхнего угла изображения. То есть сервер Web способен не только определить факт щелчка по кнопке, но и узнать координаты указателя. К сожалению, подобная обработка из сценариев JavaScript не предусмотрена.

Поле выбора файлов file

Управляющие элементы этого типа позволяют пользователям выбирать файлы, содержимое которых может передаваться приложениям сервера Web вместе с данными формы. Поля выбора файлов состоят из однострочного поля для ввода текста (имени файла) и присоединенной кнопки «Обзор… » (рисунок 3.1).

Рисунок 3.1. Форма с полем для выбора файлов



Пользователю предлагается вручную указать полное имя файла или нажать кнопку «Обзор…» для его выбора с помощью стандартного диалога текущей операционной системы.

Поле для выбора файлов также устанавливается с помощью оператора `<INPUT>`.

Пример 3.1. Форма для загрузки файлов

```
<INPUT TYPE="file"
  NAME="Имя_поля_для_выбора_файла"
  SIZE="Размер_текстовой_части">
```

В качестве примера использования поля для выбора файлов приведем исходный текст страницы, показанной на рисунке 3.1:

```
<HTML>
<HEAD>
<TITLE>Загрузка файлов</TITLE>
</HEAD>
<BODY>
<H1>Загрузка файлов</H1>
<P><B>Укажите файл для загрузки на сервер</B>

<FORM ENCTYPE="multipart/form-data"
       METHOD="post" ACTION="application.cgi">
<INPUT TYPE="file" NAME="file" SIZE="30">
<P>
```

```
<INPUT TYPE="submit" VALUE="Загрузить на сервер">
</FORM>

</BODY>
</HTML>
```

Обратите внимание, что в описании самой формы указано нестандартное значение параметра ENCTYPE и в качестве метода отправки используется метод post. Кнопка «Загрузить на сервер» является кнопкой типа submit и при ее нажатии файл отправляется серверному приложению, имя которого в приведенном примере обозначено как application.cgi. Обработку принятого файла на стороне сервера мы рассматривать не будем, так как это выходит за рамки настоящего пособия.

Скрытое поле hidden

Скрытое поле hidden используется для хранения и пересылки данных, которые могут использоваться серверными приложениями, но которые не следует отображать на экране. Добавление таких полей производится оператором <INPUT> и никак не сказывается на внешнем виде страниц HTML:

```
<INPUT TYPE="hidden"
  NAME="Имя"
  VALUE="Значение">
```

Имя и Значение, как правило, устанавливаются серверным приложением Web и считываются при повторном обращении пользователя к серверу. Это позволяет сохранять сеансы пользователей и текущие параметры работы с приложением.

Кнопки с графическими изображениями

Спецификация языка HTML поддерживает возможность использования кнопок с графическими изображениями на своей поверхности. Такие кнопки создаются оператором <BUTTON> и они могут соответствовать стандартным кнопкам button, submit и reset, создаваемым оператором <INPUT>.

Описание кнопок оператора <BUTTON> производится следующим образом:

```
<BUTTON TYPE="Тип_кнопки"
  NAME="Имя_кнопки"
  VALUE="Значение">Содержимое_кнопки</BUTTON>
```

Параметр TYPE задает тип кнопки, который может принимать одно из значений: button, submit и reset. Имя кнопки, задаваемое параметром NAME стандартным образом используется приложениями, а значение параметра VALUE предназначено только для обработки сценариями и на внешний вид кнопки не влияет.

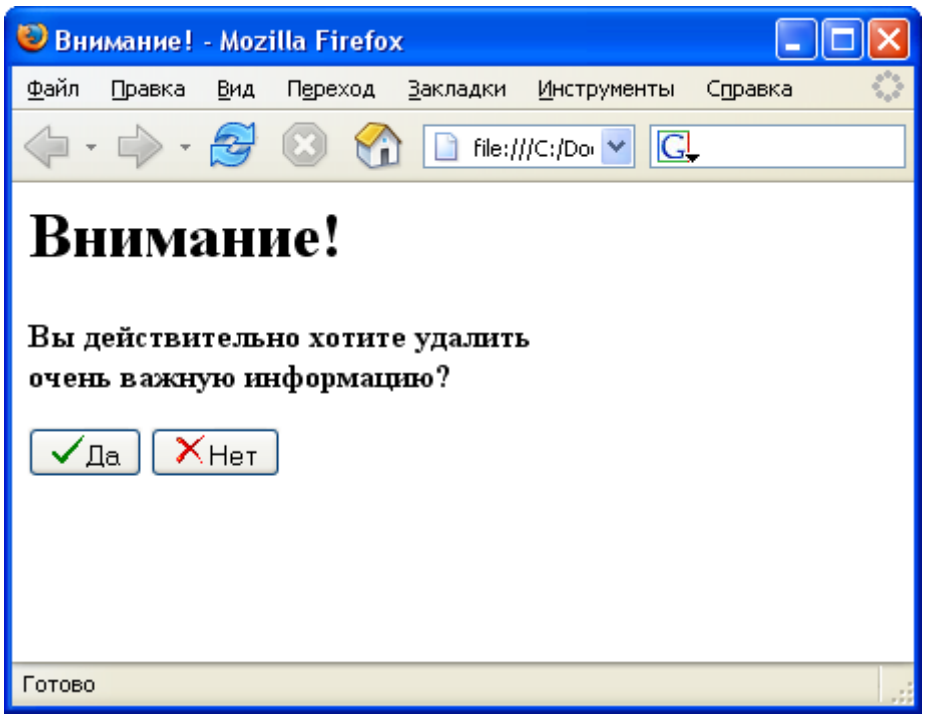
Содержимое кнопки (изображение на поверхности) задается между открывающим и закрывающим тегами <BUTTON> и может содержать как текст, так и теги <IMG> для установки картинок.

Взаимодействие этих кнопок со сценариями JavaScript полностью аналогично взаимодействию стандартных кнопок, созданных оператором <INPUT> и в примере 3.2 мы отобразим только особенности включения кнопок с графическими изображениями в документы HTML.

Пример 3.2. Использование кнопок с графическими изображениями

Рассмотрим документ, в котором пользователю предлагается вопрос и два варианта ответа («Да», «Нет»), выбрать которые можно с помощью кнопки с графическими изображениями. Внешний вид документа представлен на рисунке 3.2.

Рисунок 3.2. Использование кнопок с графическими изображениями



Исходный текст данного документа приведен ниже:

```
<HTML>
<HEAD>
<TITLE>Внимание!</TITLE>
```

```
</HEAD>
<BODY>
  <H1>Внимание!</H1>
  <P><B>Вы действительно хотите удалить<BR>
        очень важную  информацию?</B>

  <FORM NAME="frm">
    <BUTTON TYPE="button" NAME="yes"><IMG
      SRC="yes.gif">Да</BUTTON>
    <BUTTON TYPE="button" NAME="no"><IMG
      SRC="no.gif">Нет</BUTTON>
  </FORM>

</BODY>
</HTML>
```

Как видно из приведенного текста, графические изображения хранятся в файлах `yes.gif` и `no.gif`. Эти файлы стандартным образом включены вовнутрь парных тегов `<BUTTON>` с помощью оператора `<IMG>`. Кнопки содержат также и текстовые надписи («`Да`», «`Нет`»), которые оформлены простым текстом также внутри парного тега `<BUTTON>`.

Еще раз отметим, что обработка нажатия этих кнопок из сценария JavaScript производится стандартным способом и описано нами выше.

Раздел 4. Использование cookie для хранения данных форм

Cookie – это небольшая порция информации, которая ассоциирована с документом HTML и долговременно хранится браузером на клиентском компьютере. Как правило, cookie создается некоторым приложением для хранения и последующего использования различных параметров и настроек.

Основное, для чего создавалась технология cookie, – это поддержка сеансовых соединений серверных приложений (систем электронной почты, форумов, Интернет-магазинов), долговременное хранение различных настроек активных документов Web. В этом случае cookie задается серверным приложением и пересылается клиенту параллельно с запрошенной страницей HTML. Cookie сохраняется браузером и пересылается обратно всякий раз при обращении пользователя к страницам и приложениям сервера. Подобный обмен информацией позволяет поддерживать сеанс связи (в cookie может задаваться идентификатор сеанса, учетная запись пользователя и т.п.), хранить настройки документов (если, например, в cookie задан фоновый цвет страницы, то он всякий раз может анализироваться сервером и устанавливаться для каждой страницы, отсылаемой пользователю).

В нашем случае речь пойдет о локальной обработке cookie из сценариев JavaScript. Для нас важно, что cookie – это единственный способ, который позволяет сценариям JavaScript организовывать долговременное **хранение** данных на жестких дисках клиентских компьютеров. Кроме того, cookie можно использовать и для **передачи** данных между сценариями JavaScript **разных** страниц HTML.

Проще всего представить себе cookie как набор строковых параметров, каждый из которых имеет имя и значение. Сценарий JavaScript может создавать cookie для документа HTML, определяя в нем параметры и задавая для них произвольные значения. После создания такой набор параметров становится принадлежностью данного конкретного документа HTML и может быть проанализирован, изменен или удален сценарием JavaScript.

Следует заметить, что не стоит рассчитывать на то, что cookie – это очень надежный и долговременный способ хранения информации. Для каждого параметра, как правило, указывается срок, по истечении которого это параметр следует удалить. Кроме того, информация может быть удалена вследствие нехватки выделенного места или каких-либо других лимитов. Отмечается, что браузер имеет следующие ограничения:

- всего может храниться до 300 значений cookie;
- каждый cookie не может превышать 4 килобайт;
- с одного сервера может храниться до 20 значений cookie.

Если ограничение 300 или 20 превышается, то удаляется первая по времени запись. При превышении 4 килобайт – излишняя информация обрезается.

Обработка cookie из сценариев JavaScript

Вся обработка данных cookie из сценариев JavaScript производится с помощью свойства `cookie` объекта `document`. Это свойство имеет строковый тип и вся информация cookie записана парами «`имя=значение`», разделенными символом «`;`». При записи cookie, к этой информации можно добавить время истечения срока хранения данных, а также другую дополнительную информацию.

Не вдаваясь в детали организации хранения данных в cookie-файлах, приведем три функции, которые удобно использовать в своих сценариях для работы с параметрами cookie.

Создание cookie - addCookie

Данная функция позволяет сценариям JavaScript задавать значения cookie для страниц HTML. Значения, заданные этой функцией, будут

распространяться **на все страницы**, расположенные в одной папке с тем документом, в котором они были установлены.

```
function addCookie(szName,szValue,dtDaysExpires)
{
  var dtExpires = new Date();
  var dtExpiryDate = "";

  dtExpires.setTime(dtExpires.getTime() +
    dtDaysExpires * 24 * 60 * 60 * 1000);

  dtExpiryDate = dtExpires.toGMTString();

  document.cookie = szName + "=" +
       escape(szValue) + "; expires=" + dtExpiryDate;
}
```

Функция `addCookie` получает три параметра. Через параметр `szName` передается имя параметра, хранящегося в cookie. Параметр `szValue` определяет значение этого параметра cookie. Параметр `dtDaysExpires` – количество дней, через которое созданный cookie необходимо удалить.

Например, в следующей строке создается cookie с именем `Count`, значением `0`, причем через `10` дней браузер автоматически удалит этот cookie:

```
addCookie("Count","0",10);
```

Заметим, что если с помощью функции `addCookie` задать некоторое значение с именем, которое уже используется, то такая ситуация не считается некорректной. В этом случае произойдет обновление информации: будет записано новое значение существующего параметра, а также заново установлена дата истечения срока хранения cookie.

Получение значения cookie - findCookie

Ниже приведен текст функции, позволяющей прочитать значение ранее созданного cookie.

```
function findCookie(szName)
{
  szName+="=";
  var szRet = "";
  var nEndPosition = 0;
  var szCookieString = document.cookie;

  var nStartPosition = szCookieString.indexOf(szName);
```

```
  if (nStartPosition >= 0)
  {
    nStartPosition += szName.length;

    nEndPosition =
           szCookieString.indexOf(";",nStartPosition);

    if(nEndPosition < nStartPosition)
      {nEndPosition = szCookieString.length};

    szRet = unescape(szCookieString.substring(
                       nStartPosition,nEndPosition));
  }

  return szRet;
}
```

Данная функция позволяет записать в текстовую переменную значение параметра cookie с заданным именем. Например, значение заданного в прошлом примере параметра `Count` можно определить следующим образом:

```
var szCountValue = findCookie("Count");
```

С помощью этой функции можно также проверить, установлен ли для данного документа cookie с заданным именем:

```
if(findCookie("Count") == "")
{
  // cookie с именем Count не установлен
}
else
{
  // cookie с именем Count установлен
}
```

Удаление cookie - removeCookie

Функция `removeCookie` удаляет ранее созданные cookie. Ее работа проста и оригинальна: чтобы удалить cookie, надо указать уже прошедшую дату удаления. В данной функции для cookie указывается, что удалить его следовало одну миллисекунду назад.

```
function removeCookie(szName)
{
  var dtExpires = new Date();
  dtExpires.setTime(dtExpires.getTime() - 1);
```

```
  document.cookie = szName + "=" +
                  "; expires=" + dtExpires.toGMTString();
}
```

Примеры использования cookie

Пример 4.1. Хранение персональных настроек

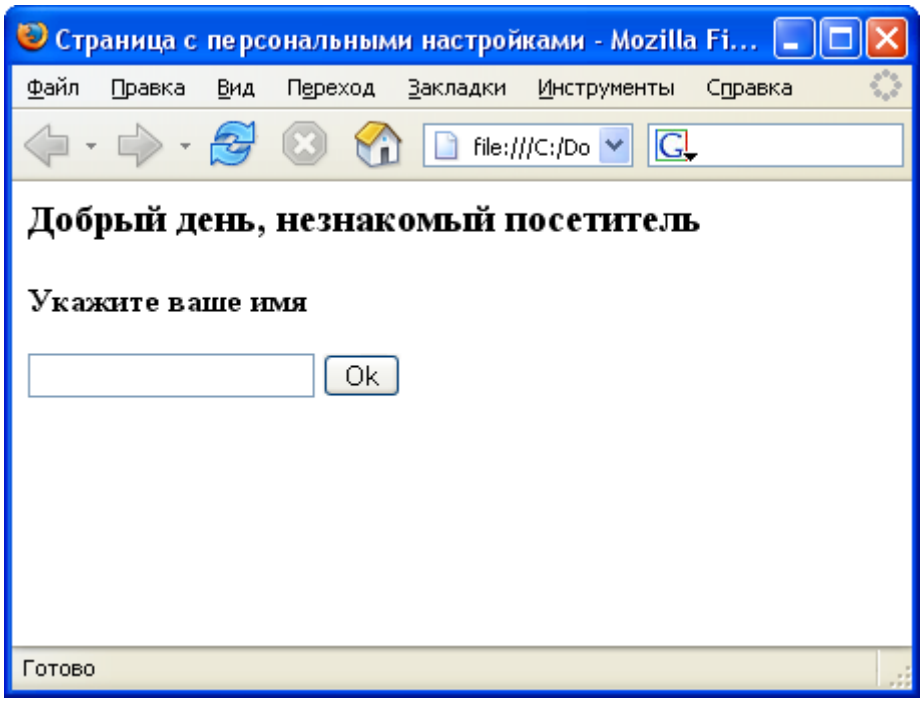
В качестве примера рассмотрим страницу, которая запрашивает имя посетителя и при последующих обращениях к себе использует это имя для подготовки строки с приветственной фразой.

```
<HTML>
<HEAD>
<TITLE>Страница с персональными настройками</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function addCookie(szName,szValue,dtDaysExpires)
{
... (Текст функции. Приведен выше.)
}

function findCookie(szName)
{
... (Текст функции. Приведен выше.)
}

function Complete()
{
  addCookie("username", document.frm.username.value, 365);
  window.location.reload();
}

// -->
</SCRIPT>
</HEAD>
<BODY>
<H3>Добрый день,

<SCRIPT LANGUAGE="JavaScript">
<!--
  var szUserName = findCookie("username");
  if (szUserName == "")
    {document.write("незнакомый посетитель"); }
  else
    {document.write(szUserName);};
//-->
</SCRIPT>
```

```
</H3>
<P><B>Укажите ваше имя</B>
<FORM NAME="frm">
  <INPUT TYPE="text" NAME="username">
  <INPUT TYPE="button" VALUE="Ok" onClick="Complete();">
</FORM>
</BODY>
</HTML>
```

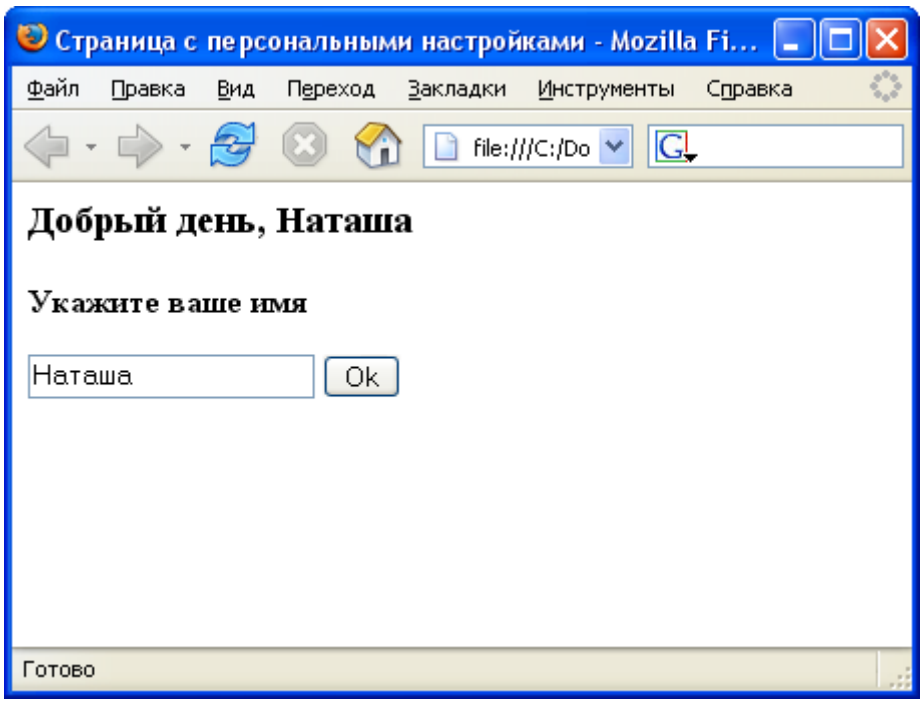
При первом обращении к документу, в окно браузера выводится фраза «Добрый день, незнакомый посетитель» и пользователю предлагается ввести свое имя (рисунок 4.1):

Рисунок 4.1. Страница при первом обращении



Если ввести свое имя и нажать кнопку «Ok», то страница будет перезагружена и приветственная фраза будет изменена в соответствии с указанными данными (рисунок 4.2):

Рисунок 4.2. Страница после указания персональных настроек



Если страницу закрыть, а потом открыть снова (возможно, через несколько дней), то информация пользователя будет восстановлена и в окне браузера появится надпись «Добрый день, Наташа».

Рассмотрим структуру представленного документа более подробно. В заголовке документа расположено три функции: `addCookie`, `findCookie` и `Complete`. Первые две из них рассмотрены нами выше и в тексте примера их описание не приводится.

В теле документа также присутствует фрагмент сценария JavaScript:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  var szUserName = findCookie("username");
  if (szUserName == "")
    {document.write("незнакомый посетитель"); }
  else
    {document.write(szUserName);};
//-->
</SCRIPT>
```

Этот сценарий с помощью функции `findCookie` обращается к cookie открытого документа и пытается определить значение параметра `username`:

```
  var szUserName = findCookie("username");
```

Если такой параметр в cookie не представлен, то считается, что пользователь первый раз посетил страничку и в окно браузера выводится

фраза "незнакомый посетитель", являющаяся завершением приветственной строки.

```
if (szUserName == "")
  {document.write("незнакомый посетитель"); }
```

Если же параметр был определен, то в завершение приветственной строки выводится его значение (имя пользователя):

```
else
  {document.write(szUserName);};
```

Для ввода имени пользователя документ HTML содержит форму с текстовым полем и кнопкой «Ok». Сохранение введенных данных в cookie производится в функции `Complete()`, назначенной обработчиком события `onClick` кнопки «Ok»:

```
function Complete()
{
  addCookie("username", document.frm.username.value, 365);
  window.location.reload();
}
```

Как видим, функция состоит всего из двух строк. Первой строкой с помощью функции `addCookie` задается параметр `username` со значением `document.frm.username.value` (строка, введенная в текстовое поле) и временем хранения в `365` дней.

Второй строкой вызывается метод `reload()` объекта `location` окна браузера. Вызов этого метода приводит к перезагрузке текущего документа. Перезагрузка производится после установки нужных значений cookie, в результате чего введенное пользователем имя сразу отображается в окне браузера.

Пример 4.2. Передача данных cookie другому документу

Как было отмечено выше, установленные значения cookie с помощью функции `addCookie()` распространяются на все документы, расположенные в одной папке с той страницей, где они были установлены[1]. Это позволяет организовать передачу данных между сценариями JavaScript в разных документах HTML. Все, что для этого требуется, в одном документе установить необходимые cookie, а в другом их прочитать.

В качестве примера создадим документ, который использует cookie, созданные в документе из примера 4.1., для подготовки своего содержания.

```
<HTML>
```

[1] Можно настроить и другое поведение cookie. Об этом подробно написано в [9] (Андрей А.Аликберов. Что такое cookies и как с ними работать).

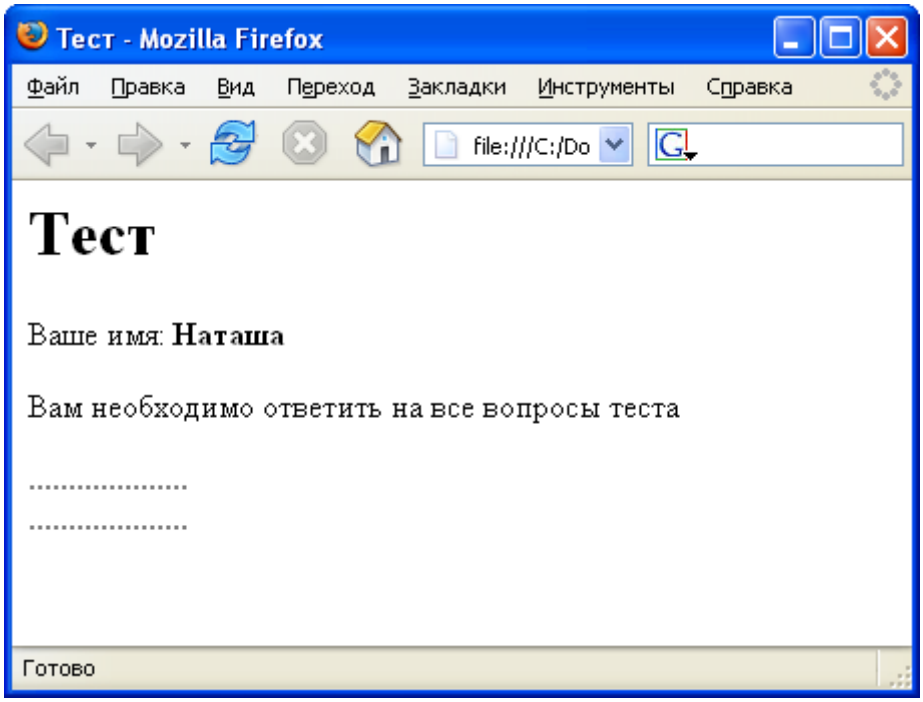
```
<HEAD>
<TITLE>Тест</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function findCookie(szName)
{
... (Текст функции. Приведен выше.)
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<H1>Тест</H1>
<P>Ваше имя:
<B>
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.write(findCookie("username"));
//-->
</SCRIPT>
</B>
<P>Вам необходимо ответить на все вопросы теста
<P><B>.....................</B><BR>
<B>.....................</B>
</BODY>
</HTML>
```

Еще раз отметим, что для правильной работы данный документ должен находиться в той же папке, что и документ из примера 4.1. В этом случае с помощью сценария будет определено имя пользователя, записанное в cookie с именем `username`:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.write(findCookie("username"));
//-->
</SCRIPT>
```

Страница будет выглядеть так, как это показано на рисунке 4.3.

Рисунок 4.3. Страница, подготовленная на основе данных, указанных в форме другой страницы



Раздел 5. Внешнее оформление полей и органов управления

В настоящем пособии мы рассматривали обработку сценариями JavaScript данных, введенных пользователем с помощью различных полей и органов управления, доступных для использования на страницах HTML. Внешнему оформлению этих элементов нами внимания уделялось совсем немного так как это, вообще говоря, выходит за пределы нашей тематики. Тем не менее, чтобы восполнить обозначенный пробел, приведем некоторые замечания и примеры, которые помогут более тонко настроить внешний вид и способы взаимодействия с пользователем активных элементов Web-страниц.

Во-первых, следует отметить, что элементы форм поддерживают многие параметры, которые ранее нами не указывались. Эти параметры, как правило, доступны и из сценариев JavaScript стандартным способом через соответствующие свойства объектов. Перечень наиболее важных из дополнительных параметров приведен ниже:

`TITLE`
Всплывающая подсказка, которая выводится при наведении на элемент курсора мыши. Пример использования:

```
<INPUT TYPE="text" NAME="login" TITLE="Укажите ваше имя">
```

READONLY

Этот параметр определяет, может ли пользователь изменять содержимое управляющего элемента. Если он установлен, то изменение значения управляющего элемента невозможно. Например, пользователь может читать и копировать значения текстовых полей, но не может их изменять.

DISABLED

Установка этого атрибута означает, что управляющий элемент является «отключенным» и все действия с ним блокируются. Например, можно блокировать кнопку отправки анкетных данных до полного заполнения всех полей формы.

TABINDEX

Числовой параметр, который определяет последовательность получения фокуса элементами при переходе с помощью клавиатуры. Переход производится от элементов с наименьшим значением параметра TABINDEX до элементов с наивысшим значением. Если параметр не указан или значения этого параметра для нескольких элементов одинаково, то последовательность перехода определяется по их положению на странице HTML.

ACCESSKEY

Назначает для элемента клавишу быстрого доступа. Нажатие этой клавиши (в сочетании с нажатой клавишей **Alt**) равносильно выбору данного элемента мышью. Например, если для текстового поля назначена клавиша L:

```
<INPUT TYPE="text" NAME="login" ACCESSKEY="L">,
```

то при нажатии пользователем сочетании клавиш **Alt+L** произойдет переход фокуса ввода указанному полю.

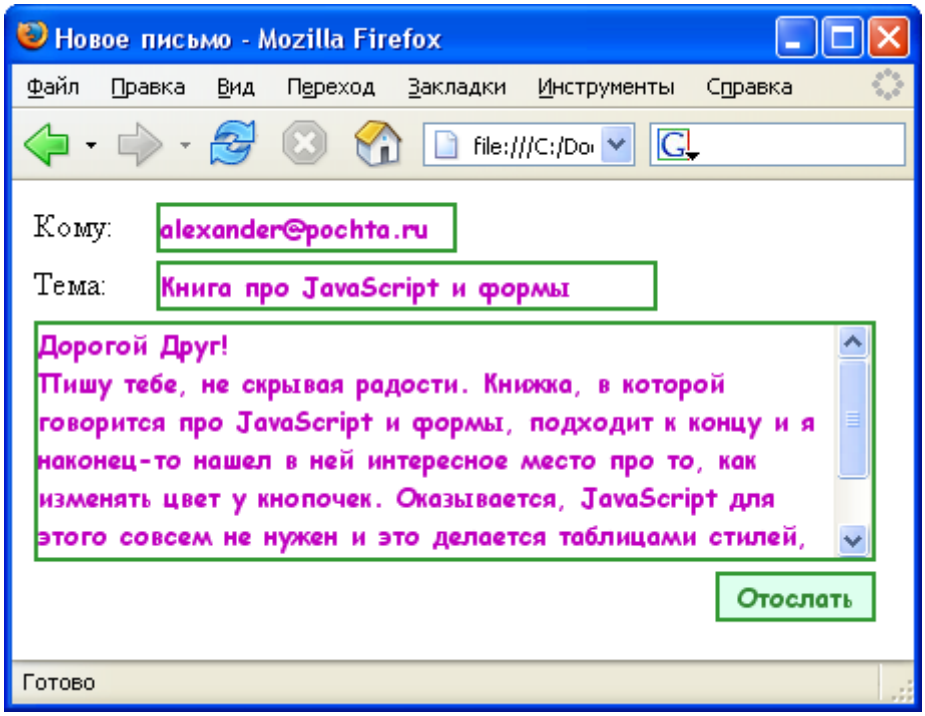
Во-вторых, в настоящее время самый богатый арсенал по внешнему оформлению различных элементов Web-страниц (в том числе – полей и органов управления форм) обеспечивают каскадные таблицы стилей. Подробное изучение стилевых параметров и особенностей их использования – тема для отдельного разговора. В нашем пособии мы ограничимся лишь одним простым примером, который поможет разобраться как в базовых стилевых параметрах (определение размеров, параметров границ, шрифтов, цветов), так и в способах взаимодействия таблиц стилей и элементов форм.

Пример 5.1. Использование таблиц стилей для оформления внешнего вида полей и органов управления форм

Рассмотрим форму, предназначенную для написания электронного письма. Форма содержит однострочные текстовые поля («Кому», «Тема»), многострочное текстовое поле (текст письма) и кнопку «Отослать». Для всех

полей указаны нестандартные параметры внешнего оформления (рисунок 5.1.)

Рисунок 5.1. Форма с нестандартными параметрами внешнего оформления



Ниже приведен исходный текст данной страницы. Обратите внимание, что в таблицах стилей указаны даже такие параметры текстовых полей, как их размер. Параметры `SIZE`, `ROWS`, `COLS` в описании текстовых полей данной страницы не используются.

```
<HTML>
<HEAD>
<TITLE>Новое письмо</TITLE>
<STYLE>
input, textarea {
  border-right: #339933 2px solid;
  border-top: #339933 2px solid;
  border-left: #339933 2px solid;
  border-bottom: #339933 2px solid;
  font-size: 10pt;
  font-family: Comic Sans MS;
  font-weight: bold;
  color: #c000c0;
  background-color: #ffffff;
}
email {width: 150px;}
subj {width: 250px;}
letter {
  width: 420px;
```

```
  height: 120px;
}
send {
  background-color: #ddffee;
  color: #338833;
}
</STYLE>
</HEAD>
<BODY>
<FORM NAME="newletter" METHOD="post" ACTION="sendmail.cgi">
<TABLE>
<TR>
  TD>Кому:</TD>
  <TD><INPUT TYPE="text" NAME="email" CLASS="email"></TD>
</TR>
<TR>
  <TD>Тема:</TD>
  <TD><INPUT TYPE="text" NAME="subj" CLASS="subj"></TD>
</TR>
<TR>
  <TD colspan="2"><TEXTAREA NAME="letter"
                        CLASS="letter"></TEXTAREA></TD>
</TR>
<TR>
  <TD colspan="2" align="right"><INPUT TYPE="submit"
                    VALUE="Отослать" CLASS="send"></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Заключение

Итак, нами рассмотрены вопросы использования форм в активных страницах Web, их обработка сценариями JavaScript. Разумеется, настоящее пособие не является исчерпывающим и охватывающим все аспекты данных технологий. Технологии Web очень быстро меняются и всегда можно обнаружить новые, ранее не известные возможности. В этой связи хочется предостеречь читателя от неприятной вероятности стать заложником этих действительно новых (или наоборот – очень старых и забытых) возможностей. Не все браузеры способны в полной мере поддерживать новейшие разработки в области Web-технологий, а от старых и малопопулярных технологий могут отказываться. Выход один: хорошо знать официальные спецификации и рекомендации их разработчиков (например, [8]) и тестировать разрабатываемые Web-документы в разных браузерах, наиболее популярных в настоящее время.

Литература

1. Сергеев А.Н. JavaScript: основы построения, базовые конструкции. Дидактический практикум. – Волгоград: Перемена, 2005. – 44 с.

2. Фролов А.В., Фролов Г.В. Сценарии JavaScript в активных страницах Web: Библиотека системного программиста, том 34. – Москва: Издательство Диалог-МИФИ, 1998. – 284 с.

3. Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2002. – 272 с.

4. Вайк Аллен и др. JavaScript в примерах: Пер. с англ./Ален Вайк и др. – К.: Издательство «ДиаСофт», 2000. – 304 с.

5. Дмитриева М.В. JavaScript: простые сценарии: Заочная школа современного программирования. Занятие 1: Учебное пособие. – СПб.: Издательство ЦПО «Информатизация образования», 2003. – 27с.

6. Стефан Кох. Введение в JavaScript. – http://www.citforum.ru/internet/koch/tutorial.htm (12 янв. 2005)

7. Павел Храмцов. Практическое введение в программирование на JavaScript. – http://www.citforum.ru/internet/js_tut/index.shtml (12 янв. 2005)

8. Спецификация HTML 4.0. Рекомендация W3C. Автор перевода Юлия Поданева. – http://www.citforum.ru/internet/html40/cover.html (12 янв. 2005)

9. Андрей А.Аликберов. Что такое cookies и как с ними работать. – http://www.citforum.ru/internet/html/cookie.shtml (12 янв. 2005)

10. Фролов А.В., Фролов Г.В. Сценарии JavaScript в активных страницах Web: Библиотека системного программиста, том 34. – http://proglib.ru/detail_book.asp?id=140 (12 янв. 2005)

11. Сергеев А.Н. Система уроков по JavaScript. – http://www.fizmat.vspu.ru/books/js/ (12 янв. 2005)