В. В. ЯНЦЕВ

JAVASCRIPT
ГОТОВЫЕ ПРОГРАММЫ

Учебное пособие

· САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР ·
· 2021 ·

УДК 004
ББК 32.973я73

Я 65 **Янцев В. В.** JavaScript. Готовые программы : учебное пособие для вузов / В. В. Янцев. — Санкт-Петербург : Лань, 2021. — 200 с. : ил. — Текст : непосредственный.

ISBN 978-5-8114-6873-7

Создавая сайт, программист вынужден решать множество задач. Нужно настроить меню, продумать работу с фотографиями, создать форму для отправки сообщений, сделать удобную систему редактирования страниц, адаптировать сайт к просмотру на мобильных устройствах. Книга предлагает множество уже готовых решений для самых разнообразных проектов. Вам не придется создавать код с «нуля» — некоторые примеры нужно только немного адаптировать под свои разработки, другие легко внедрить, вообще ничего не меняя. Рекомендовано в качестве дополнительной литературы для студентов вузов, обучающихся по направлению «Информатика и вычислительная техника».

УДК 004
ББК 32.973я73

Обложка
П. И. ПОЛЯКОВА

Оглавление

Предисловие, которое обязательно надо прочитать

Здравствуй, уважаемый читатель! Как и всякий автор, в предисловии я хочу объяснить, почему и для чего была написана эта книга.

Думаю, не ошибусь, если предположу, что ее открыл коллега-программист. Неважно – начинающий или опытный. И конечно, у него возникает вопрос: а почему так важно не пропустить вводную часть? Дело в том, что в предисловии приведена полезная и нужная информация, а также ряд дельных советов.

Приступая к написанию книги, я предполагал, что моими читателями будут в первую очередь web-программисты, уже знакомые с HTML и JavaScript по книгам из разряда «для любопытных» или «для чайников» (да не обидится на меня читатель). Часто на первых порах такой программист испытывает сложности с переходом от теории к практике. Книга, предлагаемая вашему вниманию, призвана облегчить этот переход.

Начиная создавать первые сайты, web-программисты сталкиваются с необходимостью решать множество пока еще непривычных задач. Нужно настроить меню, продумать работу с фотографиями, создать форму для отправки сообщений, сделать удобную систему редактирования страниц, адаптировать сайт к просмотру на мобильных устройствах. И, наверное, каждый замышляет написать программу для какой-нибудь забавной интернет-игры.

На страницах этой книги я постарался предложить читателям различные способы решения стоящих перед ними задач. Как следует из названия, перед вами сборник готовых программ. Конечно, они не охватывают все возможные ситуации. Здесь предложены решения для наиболее типичных случаев.

Главными действующими «лицами» будут, естественно, JavaScript и HTML. Но некоторые примеры нуждаются в использовании серверных программ. Так, отправляя сообщение из формы, вы передаете ее содержимое на сервер специальной программе, которая занимается проверкой и сортировкой входящих данных, их записью в файл или отправкой по почте администратору сайта. Такие программы пишутся на других языках. Когда в 2003 году я начинал заниматься программированием, самым популярным языком серверных программ был Perl. Затем лидерство захватил PHP. Теперь в моде Python. Хотя я знаком со всеми тремя, лично у меня наибольшие симпатии вызывает PHP. Поэтому скрипты, помогающие иллюстрировать примеры, приведенные в книге – отправку формы, программу для мобильных устройств и загрузку бесконечной ленты сообщений (как в социальных сетях) – написаны на PHP. Впрочем, вам не обязательно строго следовать по моим стопам – для своих сайтов вы можете использовать скрипты, написанные на любом подходящем языке. Главное – результат.

Вообще, вспоминая былые времена, нужно сказать, что нынешним программистам стало гораздо легче: все современные браузеры почти одинаково обрабатывают HTML и JavaScript. Раньше специалистам, создающим тот или иной код, часто приходилось писать отдельные фрагменты для разных браузеров. Даже простые программы иногда разрастались до таких размеров, что их

было сложно читать. Недаром умение «ваять» кроссбраузерную верстку оценивалось как показатель мастерства и профессионализма.

Я уже говорил, какими соображениями руководствовался при написании книги. Некоторые читатели могут мне возразить: зачем такая книга, если кучу полезных программ на JavaScript можно найти в Интернете?

Поделюсь своим наблюдением: найти хорошую программу в сети на самом деле не так-то просто. Очень большое количество этих программ страдает рядом недостатков.

1. Приводится устаревшая информация. Страница появилась в сети много лет назад и ее содержание с тех пор не менялось. А поисковые системы по-прежнему выдают ее адрес по соответствующему запросу.

2. Непроверенная информация – этим особенно грешат разнообразные форумы. Некто задает вопрос и довольно скоро получает ответ с примером кода, нередко сопровождающимся таким комментарием: «Ну, я тут по быстрому слял на коленке». Неудивительно, что в таких примерах очень часто оказываются ошибки.

3. Предлагается неоправданно «навороченная» информация. Конкретная история. В последней главе книги есть раздел, посвященный программе, которая реагирует на изменение положения смартфона или планшета. Создавая свой пример, я захотел посмотреть, как в подобных случаях поступают другие разработчики. И обнаружил, что по многим форумам кочует один и тот же вариант, в котором неизвестный мне автор потратил на вызов функции 7 строк кода там, где была нужна всего одна.

4. Приводятся решения с использованием популярных библиотек, в первую очередь, jQuery. Слов нет, библиотека эта замечательная. Только пользоваться ею лучше, когда у вас уже есть некоторый опыт программирования на JavaScript. Иначе получится, как в случае с выпускником автошколы, которого сразу после экзаменов в ГИБДД пытаются посадить за руль гоночного болида Формулы 1. Всему свое время.

Теперь я расскажу, какие принципы были использованы при написании книги.

Главное в ее содержании – примеры кода. Все они написаны на HTML5 и проверены в валидаторе Консорциума Всемирной паутины https://validator.w3.org/. На момент написания книги отклонений от стандартов HTML5 в примерах не было.

Также все примеры были проверены в наиболее популярных браузерах. Таковыми автор посчитал Microsoft Edge, Mozilla Firefox, Opera, Google Chrome и Яндекс.Браузер (а также мобильные версии двух последних). В перечисленных браузерах все примеры работали корректно.

Еще один важный момент – у книги есть сайт поддержки: http://bookjs.ru/. На его страницах вы можете увидеть программы в действии, а также посмотреть и скопировать их код.

Создавая примеры, я старался построить их таким образом, чтобы любой читатель мог применить этот код в своих разработках после внесения минимальных изменений или вообще ничего не меняя. В первой главе будут примеры работы с изображениями. Достаточно в тексте такой программы изменить

только один параметр – адрес графического файла – и ее можно вставлять в любую собственную HTML-страницу. А в примере с подсказками из третьей главы и вовсе не понадобятся никакие изменения.

Тем читателям, что будут экспериментировать и вносить коррективы в разметку и код, хотел бы дать несколько советов.

Чтобы сделать примеры кода в этой книге максимально простыми и понятными, из них удалены все лишние элементы разметки и не играющие важной роли стили. Оставлено только самое необходимое. Перенося код в свои страницы, вы столкнетесь с необходимостью «облагородить» их внешний вид. Тут перед вами широкое поле для творчества.

Подробнее коснемся стилей. Считается хорошим тоном отделять их от разметки. Обычно таблицы стилей помещают во внешних файлах. Этот метод подходит, если речь идет о большом наборе типовых страниц с идентичным стилевым оформлением. Когда вы создаете одностраничный сайт или проект, в котором все страницы разные, можно пойти другим путем – разместить стили непосредственно на странице. Например, в области <head> </head> или в теге, который нуждается в оформлении. В некоторых случаях последние два способа могут быть даже предпочтительнее. Если у страницы сложное стилевое оформление, проще настраивать его в одном окне. Согласитесь, что постоянно переключаться между несколькими документами – утомительное занятие. Поэтому, разрабатывая собственный сайт, можете придерживаться способа, удобного лично вам. Если же вы планируете работать в IT-компании, приучайте себя размещать таблицы стилей в отдельных файлах. Таковы корпоративные правила. Мы, создавая примеры, будем помещать стили в теге head, чтобы каждый наш листинг представлял собой единое целое.

Добавлю на тему стилей еще одно важное наблюдение: в некоторых (редких) случаях вы можете получить совершенно разные результаты, разместив одни и те же настройки в теге и в головной части документа. Заглянем на сайт поддержки книги, на страницу http://bookjs.ru/TEST/TEST.html. Посмотрите: у левой картинки атрибут style размещен в теге img, поэтому тестовая программа «слева» работает правильно. У картинки справа точно такие же настройки стиля располагаются в головной части документа внутри области head. В результате правый снимок невозможно открыть выбранным мною методом. Желающие убедиться в реальности этого примера могут посмотреть код страницы. Заставить открываться правый снимок можно, только изменив принцип работы функции.

Несколько слов о комбинировании JavaScript и HTML. В среде программистов принято отделять от разметки не только стили, но и код. Мы будем придерживаться этой традиции. Так же, как и в ситуации с таблицами стилей, наш код JavaScript будет находиться в головной части документа. Но при этом добавлю, что правилами HTML5 разрешено помещать вызовы функций и даже JavaScript-код внутри элементов. Хотя и не рекомендуется. Но можно.

Хочу дать еще один полезный совет. Позволяя клиенту выполнять какие-то действия на сайте, обязательно проверяйте, не содержит ли его функционал не предусмотренных вами возможностей. Объясню ситуацию на примере из

моей практики. Однажды я запускал Интернет-портал, в работе которого могли принимать участие зарегистрированные пользователи. Они имели возможность оставлять на своих страницах разные сообщения, а также редактировать последнее из них. Для этого предназначались две кнопки: «Добавить сообщение» и «Редактировать последнее сообщение». По моим представлениям, каждый человек должен был интуитивно понимать, что сначала надо что-то написать, а потом уже редактировать написанное. В качестве первых участников портала я пригласил нескольких своих знакомых. И тут меня ожидал сюрприз. Двое из них начали с того, что попытались отправить текст через форму «Редактировать последнее сообщение», что привело к возникновению ошибок в работе портала. Пришлось менять подход – кнопка редактирования стала появляться только после написания первого поста. Этот опыт научил меня всегда придерживаться принципа: если посетитель может что-то делать на сайте, необходимо так направлять его действия, чтобы он двигался только одним, единственно верным путем.

И, наконец, последнее соображение, которым я хотел поделиться с вами – каким редактором пользоваться при написании кода. Я долгое время создавал все программы в обычном приложении ОС Windows – Блокноте. Но однажды опробовал несколько специализированных редакторов. Мне понравился Notepad++ и с тех пор я пишу только в нем. Заметьте, я не призываю всех переходить на этот редактор. Вы можете пользоваться той программой, которая вам нравится. Но если раньше вы работали в простейшем редакторе типа Блокнота, имеет смысл подумать о переходе на специализированный. Такие редакторы гораздо удобнее: они подсвечивают код, предлагают синтаксические подсказки, позволяют менять кодировку документов, сохраняют файлы в Win-, Unix- или Mac-формате.

Вот и все, что я хотел сказать в предисловии. Настало время перейти к практике.

Глава 1
Фотографии во всем многообразии

Итак, предисловия завершены и пора приступить к делу. А начнем мы с вопроса. Из чего обычно состоит первый сайт начинающего программиста? А точнее – любая из его страниц? Скорее всего, она содержит несколько пунктов навигационного меню, текст и, конечно, графику – рисунок, фотографию или схему. Как раз о графических элементах у нас и пойдет речь в первой главе. Здесь мы поговорим о том, как размещать изображения, и о том, как ими манипулировать.

Просмотр фото

Ваш сайт будут посещать разные люди. Нужно быть готовым к тому, что все они станут вести себя по-разному. Кто-то бегло взглянет на страницу и уйдет дальше в просторы Интернета. Кто-то задержится немного дольше. А кого-то информация, представленная на странице, заинтересует всерьез.

Естественно, что этим заинтересованным читателям вы хотите представить свои идеи, соображения, творчество в самом лучшем виде. С текстом все понятно – достаточно использовать хорошо читаемый шрифт. С фотографиями или рисунками ситуация несколько иная. Чаще всего, чтобы ускорить загрузку страницы и не слишком растягивать ее, программисты вставляют в текст уменьшенную копию изображения. Но при этом необходимо дать посетителю возможность посмотреть изображение в более высоком разрешении.

Нет ничего проще! Все прекрасно осведомлены, что на рисунке, предназначенном для детального изучения, достаточно щелкнуть указателем мыши – и можно рассматривать увеличенное изображение.

Как достигается этот результат? Существуют разные способы увеличить картинку на странице сайта. Мы с вами рассмотрим два варианта.

Способ первый.

Создадим простейшую HTML-страницу. Ее код приведен ниже:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Просмотр фото</title>
</head>
<body>
</body>
</html>
```

В тело документа (между тегами **<body> </body>**) вставим маленькое изображение:

```
<img src="picture.jpg" alt="Фото">
```

Укажем в головной части документа размеры фото:

```
<style>
img {width: 100px; height: 100px;}
</style>
```

Дальше необходимо добавить к изображению ссылку, по щелчку на которой будет открываться та же картинка, но размером, предположим, 500 на 500 пикселей. Этот процесс разобьем на несколько этапов.

Первым делом «обернем» (как говорят некоторые программисты) картинку в ссылку:

```
<a href="#" id="lin"><img src="picture.jpg" alt="Фото"></a>
```

Теперь надо написать функцию, которая будет реагировать на клик мышью по маленькому фото. И вот здесь нам впервые представится случай использовать возможности JavaScript. Дело в том, что в этом языке программирования есть удобный способ открыть изображение (или целую страницу) в новом окне. Для этого используется метод **window.open()** (то есть команда на открытие нового окна). В скобках указываются необходимые программисту параметры: адрес файла, который будет загружен в новом окне, имя нового окна, его высота и ширина, возможность изменить размеры этого окна. Обратите внимание: имя окна в данном случае – параметр, необходимый для обращения к окну или его содержимому на программном уровне. Это имя не имеет никакого отношения к заголовку в теге title.

Разместим нашу программу просмотра изображения внутри области **head**. Для этого надо вписать открывающий и закрывающий теги **<script> </script>**. И уже в них добавить функцию обработки события click.

Итак, пишем:

```
<script>
window.onload=function()
{
document.getElementById("lin").onclick=function()
  {
  window.open("picture.jpg", "", "width=500, height=500");
  return false;
  };
};
</script>
```

Инструкция

```
window.onload=function()
{
…
};
```

регистрирует обработчик события

```
document.getElementById("lin").onclick=function()
  {
  …
  };
```

после окончания загрузки страницы. С этого момента метод **window.open()** становится доступен для обращения к нему:

```
window.open("picture.jpg", "", "width=500, height=500");
```

Как видите, мы указали адрес рисунка и размеры окна – они соответствуют размеру изображения. Имя окну присваивать не стали, так как в данном случае в этом нет необходимости.

Обработчик события *– это некоторая функция, выполняющая определенные манипуляции с содержимым HTML-страницы в ответ на действия пользователя. Создавая приложение, необходимо связать событие на конкретном элементе разметки с конкретной функцией. Это называется «регистрацией обработчика». Самый простой способ регистрации обработчика – указать его непосредственно в теге элемента, например, так <img src= "picture.jpg" onClick="func()">, где func() – имя функции, реагирующей на событие. Однако такой способ применять нежелательно по причинам, о которых мы говорили в Предисловии (в рассмотренном варианте код добавлен в разметку, что считается плохим стилем программирования). Другой способ регистрации обработчиков – добавление в головную часть документа инструкций, которые после загрузки страницы сообщают программе, какие функции вызываются в ответ на разные действия посетителя. Именно этот способ мы использовали в нашей первой программе и именно его будем применять в дальнейшем.*

Теперь обратите внимание на форму записи **href="#"**. В последующих примерах она попадется еще не раз. Такую «конструкцию» применяют, когда надо показать посетителю сайта, что перед ним ссылка, которую можно нажать. Только нажатие приводит не к переходу на другую страницу, а к запуску программы на JavaScript (что и требуется в нашем случае). При клике на такой ссылке страница не станет перезагружаться. Но зато в конце адресной строки браузера появится лишний символ #, что исказит реальный адрес, а сама страница прокрутится вверх. Чтобы избавится от этих нежелательных эффектов, в конце функции, обрабатывающей клик на такой ссылке, необходимо добавить строку **return false;**. Что мы и сделали:

```
document.getElementById("lin").onclick=function()
  {
  …
  return false;
  };
```

Соберем фрагменты кода в единое целое:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Просмотр фото</title>
<style>
img {width: 100px; height: 100px;}
</style>
<script>
window.onload=function()
{
document.getElementById("lin").onclick=function()
  {
  window.open("picture.jpg", "", "width=500, height=500");
  return false;
  };
};
</script>
</head>
<body>
<a href="#" id="lin"><img src="picture.jpg" alt="Фото"></a>
</body>
</html>
```

Результат выполнения этого кода виден на рисунке 1_1.

Итак, мы досконально разобрали нашу первую программу. Заинтересованный читатель, конечно, уже обратил внимание: код есть, картинка есть, а ссылка на страницу сайта с этим примером отсутствует. Почему?

Хотя приведенный выше пример абсолютно работоспособен, мы не станем применять его для увеличения изображений. Просто таким способом мы продемонстрировали самый элементарный случай использования JavaScript – открытие нового окна браузера и загрузка в него какого-либо документа.

А чем же плох просмотр картинки в новом окне? Конечно, ничего плохого в этом нет. Скорее, такой метод можно назвать не самым удобным или не самым комфортным. Открытие нового окна с изображением требует больше времени, чем просмотр того же изображения в уже существующем окне (естественно, речь идет о просмотре без перезагрузки текущей страницы). Если но-

вое окно меньше, чем размеры монитора компьютера, то, просматривая увеличенное фото, посетитель будет видеть часть исходной страницы, что не совсем хорошо с эстетической точки зрения. Если же открывать новое окно на весь экран, то фотография окажется на белом фоне, который по умолчанию создает браузер. Хорошо, если вы стремились как раз к такому результату. А если вы хотели, чтобы фон был черным? Конечно, в этом случае можно поступить хитрее – создать HTML-страницу с черным фоном и поместить на нее изображение. И уже эту страницу открывать методом **window.open()**. Но тогда вам придется создавать страницы для каждой фотографии. Если их много – вы обеспечите себя большим объемом работы. А это не просто неудобно – это очень нерационально. Кроме того, возникает вопрос: а зачем в таком случае вообще использовать JavaScript? Можно в открывающем теге **a** просто написать **href="адрес_страницы" target="_blank"** и получить тот же самый результат.

Рис. 1_1. После щелчка на уменьшенном изображении открывается новое окно, где фотография представлена в «натуральную величину»

И все же пример был важен. Повторюсь: мы увидели самый простой случай использования JavaScript. Не станем же мы для этих целей писать примитивные программы в стиле «Hello, World!». Во-первых, мы с вами взрослые люди и хотим заниматься вещами, имеющими практическое значение. А во-вторых, чтобы вывести в браузере надпись «Hello World!» никакие языки программирования не требуются (и даже не надо делать HTML-разметку страницы).

В реальной практике вы, скорее всего, будете очень редко прибегать к подобным методам – только если вам понадобится окно уменьшенного размера или если будет необходимо манипулировать содержимым нового окна.

Теперь попробуем вспомнить, а как механизм просмотра увеличенного изображения реализован на сайтах, которые нам нравятся? Там это выглядит довольно изящно: снимок открывается в родительском окне, посредине экрана, на темном полупрозрачном фоне.

Нам тоже так хочется? Тогда прибегнем ко второму способу. Посмотреть, как будет выглядеть такой вариант, можно на странице http://bookjs.ru/p1.html.

Напомню, что все демонстрационные варианты визуально «облагорожены» – к ним добавлено оформление. В коде примера, который вы увидите далее в тексте или посмотрите на сайте, большая часть оформления исключена – оставлено только самое необходимое.

Как видно из рисунка 1_2, в этот раз мы использовали изображение северной столицы России – города Санкт-Петербурга. Исходный размер фотографии – 640 на 413 пикселей. Уменьшенный вариант, представленный на рисунке 1_2, имеет размер 300 на 194 пикселя.

Кликнем на изображении. Страница изменит свой внешний вид – фон станет затемненным и полупрозрачным, а на переднем плане по центру экрана появится снимок в натуральную величину (рис. 1_3). Чтобы завершить просмотр и вернуться на страницу, щелкните мышью на изображении или на любом участке фона.

Нажав на ссылку «Код примера», вы увидите, каким способом этот вариант реализован. Ниже мы разберем его «по косточкам».

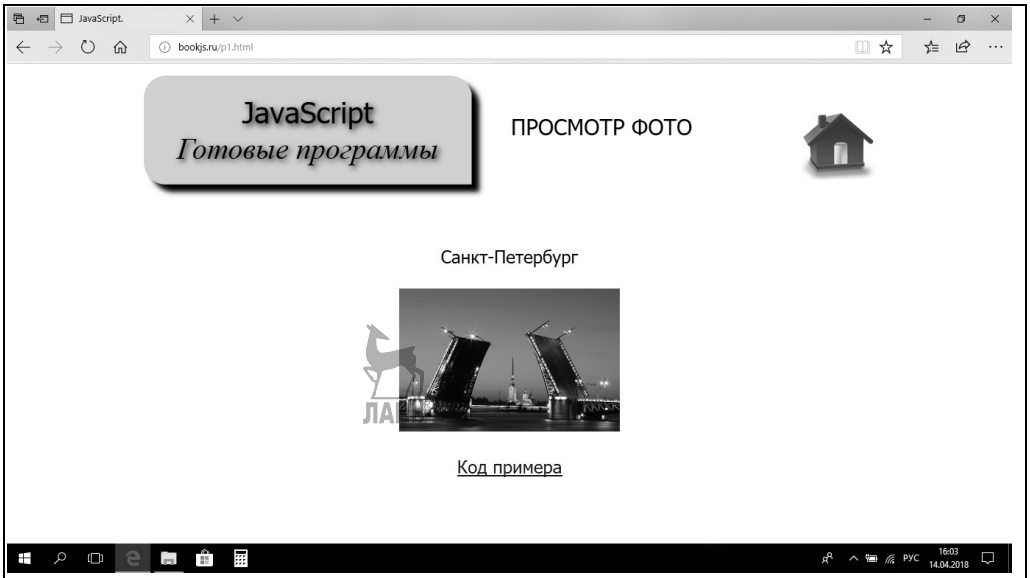

Рис. 1_2. Автор этой книги неоднократно бывал в городе на Неве и поэтому с удовольствием использовал его фотографию в данном примере

Для этого вновь создадим простейшую HTML-страницу – точно такую же, как и в первом случае. И поместим на нее фото Санкт-Петербурга:

<img src="pict/spb.jpg" class="smp" alt="Фото">

Вполне достаточно указать размер только одной стороны снимка – например, ширины – браузер сам пропорционально уменьшит высоту:

.smp {width: 300px;}

Добавим ссылку:

<img src="pict/spb.jpg" class="smp" alt="Фото">

Здесь, в отличие от первого примера, обработчик события будет запускать не команду на открытие нового окна, а функцию **look()** (смотреть), которая управляет просмотром большого изображения. Чтобы по щелчку на фоне или фотографии страница возвращалась в исходное состояние, в программу добавлена функция **del()** (от английского delete).

О содержимом функции мы расскажем далее, а пока вернемся к разметке страницы. Помимо уже сделанного, нам еще необходимо:

– создать слой с темным полупрозрачным фоном;

– добавить слой с большой фотографией;

– сделать фон и фотографию временно невидимыми.

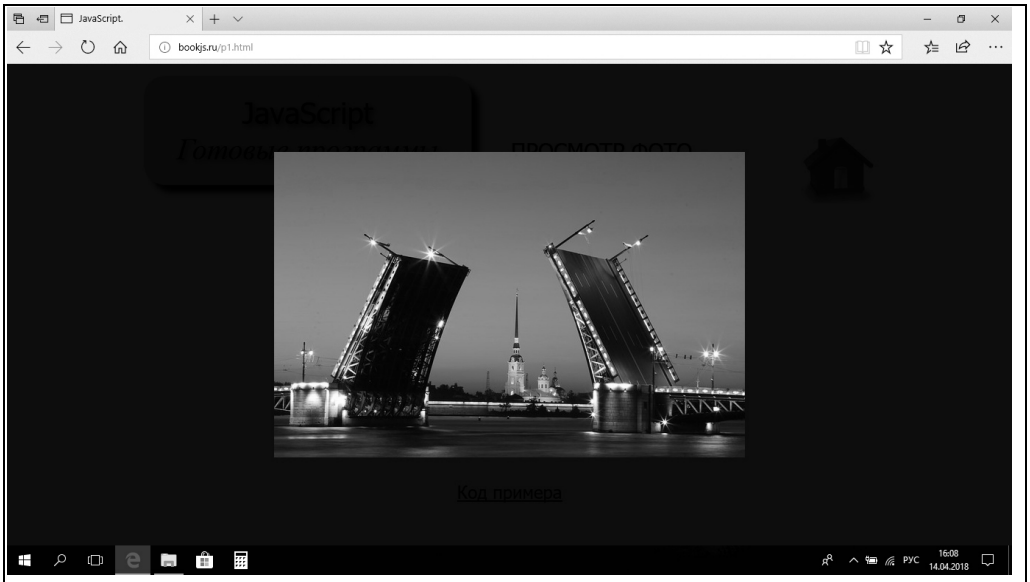

Рис. 1_3. Все как на сайтах, которые нам нравятся – темный полупрозрачный фон и фотография посередине

Начнем с фона. Можно, например, сделать темный полупрозрачный слой в файле PNG – и использовать его в качестве фона. Но не каждый программист

имеет необходимое графическое ПО для создания подобного файла. Поэтому мы пойдем другим путем – средствами, имеющимися в арсенале HTML, создадим слой

** **

с нужными характеристиками:

#bas {position: absolute; z-index: 90; left: 0px; top: 0px; width: 100%; height: 100%; visibility: hidden; background: #000000; opacity: 0.95;}

У нас получился слой черного цвета с уровнем прозрачности 0.95. Фон занимает все пространство экрана и в исходном состоянии не виден. Слой имеет собственный ID **bas** (от английского base, basis – основа). Он понадобится для управления свойствами фона на программном уровне. Индекс расположения слоя – 90 – выбран произвольно (совет: размечая на странице первый слой, присваивайте ему такое значение **z-index**, чтобы у вас была возможность маневра – давать последующим слоям более низкий или более высокий индексы – в зависимости от их положения в иерархии страницы).

Дальше надо добавить слой с большой фотографией. Логично было бы вновь использовать ** ** или вообще обойтись без этого тега, указав необходимые параметры в **style** тега **<img>**. Но из соображений совместимости с последующими примерами (где требуются более основательные подходы) мы немного усложним наш код и разместим большое изображение в теле таблицы, состоящей (пока) из одной ячейки. Вот что получится в результате:

<table id="view">
<tr><td id="tdt"><img src="pict/spb.jpg" alt="Фото"></td></tr></table>

#view {position: absolute; z-index: 91; left: 0px; top: 0px; width: 100%; height: 100%; visibility: hidden;}

#tdt {text-align: center;}

Как и фон, таблица имеет свой ID **view** (вид, просмотр), «растягивается» на весь экран и в исходном состоянии не видна. Более того – она прозрачна, тело ее не имеет фонового цвета. Поэтому при значении **visibility** равном **visible** будет виден только снимок на темном полупрозрачном фоне. **Z-index** на единицу больше, чем у слоя **bas**, благодаря чему таблица с фотографией располагается на «самом верху». Свойство **text-align** ячейки установлено **center**. За счет этого снимок центрирован по горизонтали. Параметры вертикального выравнивания в стиле ячейки не указаны. А это значит, что браузер расположит фотографию в центре также и по вертикали (так действуют по умолчанию все браузеры).

Наверное, у многих читателей уже возник вопрос: а почему понадобился отдельный фоновый слой? Не проще ли обойтись только таблицей, указав в ее настройках **background: #000000** и **opacity: 0.95**? К сожалению, такой способ нам не подойдет, так как изменится прозрачность большой фотографии (как дочернего элемента таблицы). Она станет «просвечивать», чего мы ни в коем случае не хотим.

Теперь перейдем к программе, которая управляет просмотром большой фотографии. Как и в предыдущем случае, впишем открывающий и закрывающий теги **<script> </script>** в головную часть документа. И уже в них поместим функции **look()** и **del()**:

```
<script>
window.onload=function()
{
document.getElementById("lin").onclick=look;
document.getElementById("view").onclick=del;
};

function look()
{
document.getElementById("view").style.visibility="visible";
document.getElementById("bas").style.visibility="visible";
return false;
}

function del()
{
document.getElementById("view").style.visibility="hidden";
document.getElementById("bas").style.visibility="hidden";
}
</script>
```

Первым делом произведем регистрацию обработчиков событий, которые запускают функции:

```
window.onload=function()
{
document.getElementById("lin").onclick=look;
document.getElementById("view").onclick=del;
};
```

Какую роль выполняет функция **look()**? Она делает видимыми слои **view** и **bas**. Благодаря этому посетитель после щелчка на маленькой картинке видит большое изображение на темном полупрозрачном фоне. Функция **del()** наобо-

рот делает слои **view** и **bas** невидимыми, скрывая фон и фото, если посетитель кликнет на одном из них.

Собрав разрозненные части в единое целое, мы получим следующий код:

```
<meta charset="utf-8">
<title>Просмотр фото</title>
<style>
#bas {position: absolute; z-index: 90; left: 0px; top: 0px; width: 100%;
height: 100%; visibility: hidden; background: #000000; opacity: 0.95;}
#view {position: absolute; z-index: 91; left: 0px; top: 0px;
width: 100%; height: 100%; visibility: hidden;}
#tdt {text-align: center;}
.smp {width: 300px;}
</style>

<script>
window.onload=function()
{
document.getElementById("lin").onclick=look;
document.getElementById("view").onclick=del;
};

function look()
{
document.getElementById("view").style.visibility="visible";
document.getElementById("bas").style.visibility="visible";
return false;
}

function del()
{
document.getElementById("view").style.visibility="hidden";
document.getElementById("bas").style.visibility="hidden";
}
</script>
</head>

<body>

<div id="bas"></div>
<table id="view">
<tr><td id="tdt"><img src="pict/spb.jpg" alt="Фото"></td></tr></table>

<a href="#" id="lin"><img src="pict/spb.jpg" class="smp" alt="Фото"></a>
```

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

Этот код легко адаптировать под свои нужды. Достаточно заменить адрес изображения – и его можно внедрять в свой проект.

Выбор фото

Мы опробовали простой и удобный способ увеличения размера фотографии при просмотре. Но он прост и удобен, когда речь идет о единственном снимке на странице. А если их больше? Ну хотя бы два? Как поступить в таком случае: создавать новые слои для нового снимка, добавлять новые функции просмотра и отключения большой фотографии? Нет, мы выберем более рациональный подход и создадим код, который будет использовать одни и те же слои – **view** и **bas** – для демонстрации разных изображений.

Наш рационализм начнется с того, что мы не станем писать новую программу для случая двух фотографий, а модернизируем уже имеющуюся. Попутно замечу, что изменения будут не очень большими – основа останется прежней.

Начнем с того, что поместим на страницу два снимка. В качестве образцов я выбрал фотографии Москвы – дневной и вечерней. Что получилось, видно на рисунке 1_4. Так же посмотреть действующий пример можно на странице http://bookjs.ru/p2.html.

С появлением нового изображения тело документа будет выглядеть так:

```
<div id="bas"></div>
<table id="view">
<tr><td id="tdt"><img id="photo" src="pict/point.jpg"
alt="Фото"></td></tr></table>

<div id="pic">
<a href="#"><img src="pict/mos1.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="pict/mos2.jpg" class="smp" alt="Фото"></a>
</div>
```

При внимательном изучении приведенного выше отрывка становится понятно, что структура фонового слоя и таблицы остались прежними. Изменилось содержимое тега **img** в теле таблицы. Во-первых, мы дали изображению ID, чего не делали в предыдущем примере (за ненадобностью). Во-вторых, атрибуту **src** теперь присвоен адрес **pict/point.jpg**. Это микроскопическая черная точка размером 1x1 пиксель. По стандартам HTML5 атрибуту src всегда должен быть присвоен какой-то конкретный адрес. Так как теперь в исходном состоянии ни один из двух снимков Москвы не загружен в слой **view**, мы, следуя правилам,

загружаем вместо них нейтральное миниатюрное изображение, которое не видно на темном фоне **bas** и быстро заменяется одной из двух фотографий столицы.

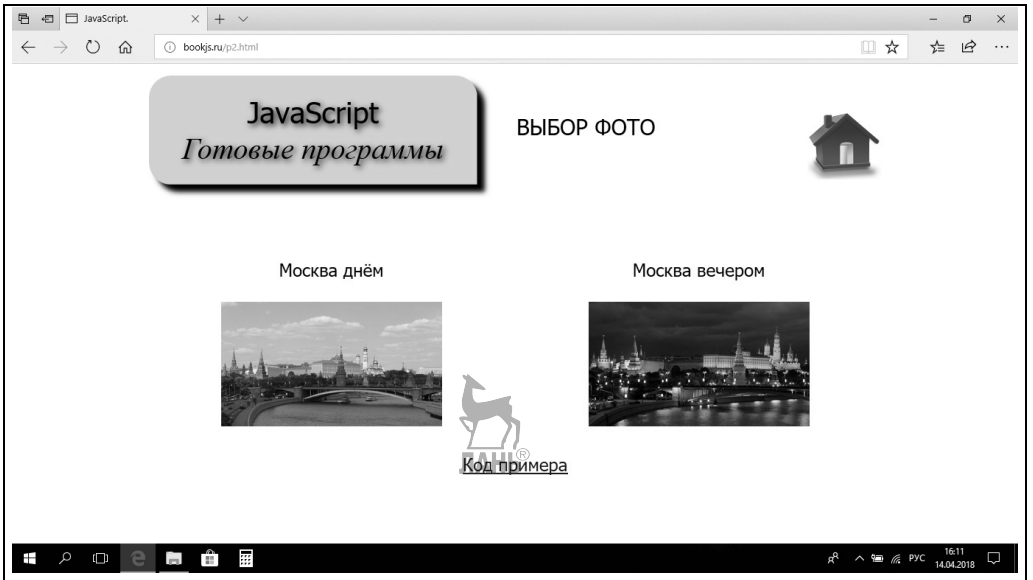

Рис. 1_4. Мы уже использовали в качестве примера фотографию Санкт-Петербурга, поэтому имеет смысл продолжить ранее выбранную линию и обратиться к другим городам России. Снимки столицы отлично подходят для иллюстрации работы нового варианта программы

Продолжая разбирать фрагмент, вы увидите в разметке контейнер **pic** с двумя новыми фотографиями. По сравнению с предыдущим случаем у ссылок нет ID, так как в новой версии программы обработка щелчка мышью происходит иначе, чем прежде:

```
window.onload=function()
{
document.getElementById("pic").onclick=look;
document.getElementById("view").onclick=del;
};
```

Мы теперь отслеживаем клики на слое **pic**. Сначала определяем, что событие click произошло на фотографии, затем узнаем ее адрес и присваиваем этот адрес атрибуту **src** элемента **img** таблицы. Соответствующее изображение загружается, после чего фон и содержимое таблицы становятся видимыми. В результате появляется увеличенное изображение дневной или вечерней Москвы. Иллюстрирует это описание рисунок 1_5.

Таким образом, каждый раз, щелкая на фотографии, мы вызываем загрузку определенного изображения. Завершая просмотр щелчком на фоне или

снимке, мы одновременно должны вновь присвоить **src** значение **pict/point.jpg**. В принципе, этого можно было бы и не делать. Но! В таком случае при низкой скорости соединения с сервером можно столкнуться с неприятным эффектом: допустим, вы посмотрели дневную Москву, закрыли фотографию и щелкнули на снимке вечернего города. Вновь запустится режим просмотра, но из-за медленной загрузки второго снимка вы некоторое время будете видеть первый. А это, согласитесь, не очень хорошо. Присваивая **src** значение **pict/point.jpg**, мы избавляемся от подобных неприятностей, ведь маленькая черная тока загружается очень быстро и совершенно не будет видна в момент медленной загрузки второго снимка. То есть мы просто будем видеть темный полупрозрачный фон до тех пор, пока не появится изображение вечерней Москвы.

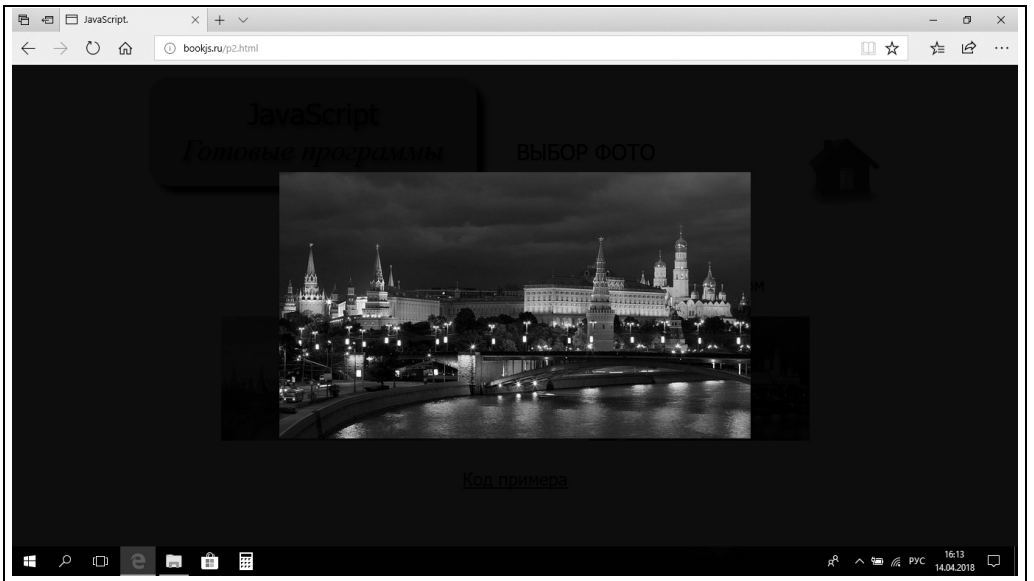

Рис. 1_5. Москва прекрасна и днем, и вечером

Новые возможности реализованы благодаря усовершенствованию функций **look()** и **del()**:

```
function look(event)
{
if(event.target.tagName=="IMG")
  {
  document.getElementById("photo").src=event.target.src;
  document.getElementById("view").style.visibility="visible";
  document.getElementById("bas").style.visibility="visible";
  }
return false;
}
```

```
function del()
{
document.getElementById("view").style.visibility="hidden";
document.getElementById("bas").style.visibility="hidden";
document.getElementById("photo").src="pict/point.jpg";
return false;
}
```

Теперь функция **look()** вызывается с аргументом, в качестве которого передается объект события **event**:

```
function look(event)
{
…
}
```

Объект события event *– это конструкция языка JavaScript, содержащая информацию о произошедшем на странице событии: перемещении мыши, щелчке на ссылке, получении элементом фокуса и многих других. Объект события имеет свойство* ***target****, указывающее, на каком элементе произошло событие. Это свойство позволяет получить доступ к дополнительной информации, например, узнать имя тега, на котором был выполнен клик, или его ID.*

Первым делом выполняется проверка, на каком элементе произошел клик:

```
if(event.target.tagName=="IMG")
```

Если этим элементом оказался один из снимков, выясняем адрес и присваиваем его атрибуту **src** тега **img** таблицы:

```
document.getElementById("photo").src=event.target.src;
```

Затем происходит демонстрация большой фотографии на темном фоне:

```
document.getElementById("view").style.visibility="visible";
document.getElementById("bas").style.visibility="visible";
```

Как и в предыдущих случаях, последняя строка функции

```
return false;
```

возвращает логическое значение «ложь», благодаря чему страница остается «на месте», а в адресной строке не появляется лишний знак #.

Когда вызывается функция **del()**, атрибуту **src** вновь присваивается значение **pict/point.jpg**:

```
function del()
{
…
document.getElementById("photo").src="pict/point.jpg";
}
```

Механизм «проявления» и «исчезновения» фона и снимка изменений не претерпел.

Завершая рассказ о новом способе работы с изображениями, приведем полный код примера:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Выбор фото</title>
<style>
#bas {position: absolute; z-index: 90; left: 0px; top: 0px; width: 100%; height:
100%; visibility: hidden; background: #000000; opacity: 0.95;}
#view {position: absolute; z-index: 91; left: 0px; top: 0px; width: 100%; height:
100%; visibility: hidden;}
#tdt {text-align: center;}
.smp {width: 300px;}
</style>

<script>
window.onload=function()
{
document.getElementById("pic").onclick=look;
document.getElementById("view").onclick=del;
};

function look(event)
{
if(event.target.tagName=="IMG")
  {
  document.getElementById("photo").src=event.target.src;
  document.getElementById("view").style.visibility="visible";
  document.getElementById("bas").style.visibility="visible";
  }
return false;
}

function del()
{
```

```
document.getElementById("view").style.visibility="hidden";
document.getElementById("bas").style.visibility="hidden";
document.getElementById("photo").src="pict/point.jpg";
return false;
}
</script>
</head>

<body>

<div id="bas"></div>
<table id="view">
<tr><td id="tdt"><img id="photo" src="pict/point.jpg"
alt="Фото"></td></tr></table>

<div id="pic">
<a href="#"><img src="pict/mos1.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="pict/mos2.jpg" class="smp" alt="Фото"></a>
</div>

</body>
</html>
```

Как и в первом случае добавим, что вы можете использовать данный код для своих разработок, просто поменяв адреса файлов.

Галерея

Разбирая предыдущий пример, вы могли обратить внимание на одно неудобство. Каждый раз, прежде чем просматривать следующую фотографию, необходимо сначала закрыть предыдущую. В случае с двумя изображениями это не критично. Когда изображений много, посетителя начнет раздражать такая непрактичная система. Поэтому в следующем примере мы избавимся от подобных неудобств.

Возьмем случай, когда на нашей странице 9 изображений. Для их просмотра мы создадим программу «Галерея». Как она будет работать? По щелчку на любом фото страница переключится в режим просмотра увеличенных снимков. Одновременно на темном фоне появятся кнопки для пролистывания фотографий вперед или назад. Нажимая их, вы можете перемещаться по галерее в любом направлении, возвращаясь при необходимости к ранее просмотренным кадрам. Пример такой галереи представлен на странице http://bookjs.ru/p3.html. Продолжая алгоритм выбора изображений, используем в примере фото различный российских городов – Москвы, Санкт-Петербурга, Казани, Волгограда и других. Что у нас получилось, видно на рисунке 1_6.

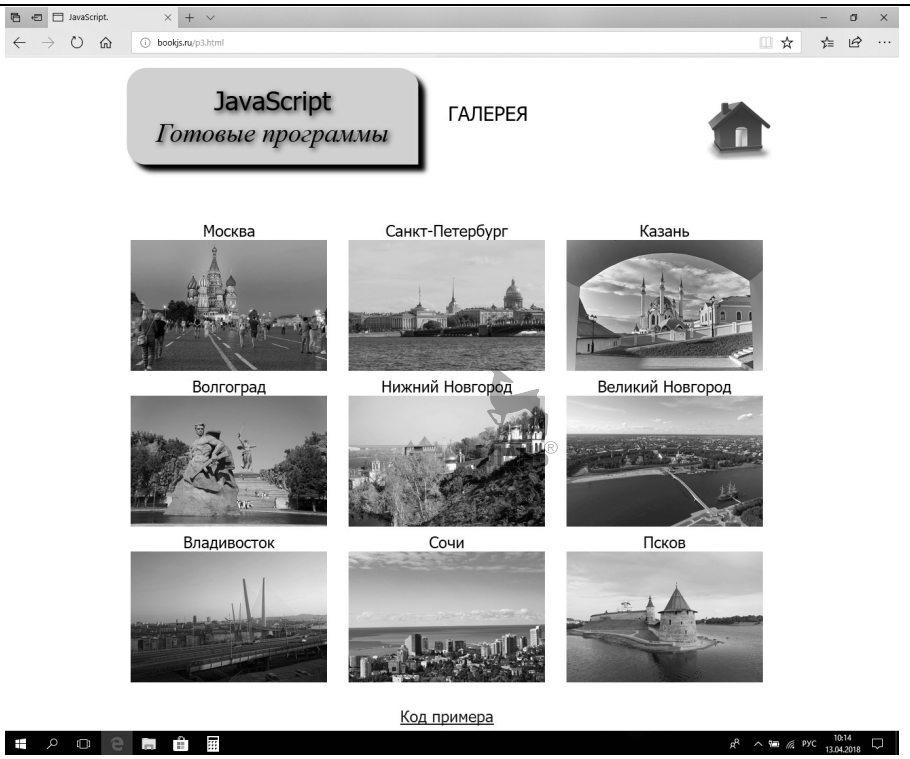

Рис. 1_6. Галерея городов России

Будет еще два новшества: теперь щелчок на темном фоне не вызовет никакого действия (клик на фото по-прежнему будет приводить к завершению просмотра) и появится отдельная кнопка для выхода из режима галереи. Почему мы отказались от завершения просмотра по щелчку на фоне, объясним в процессе разбора кода.

Добавим на страницу фотографии, разместив их рядами по 3 штуки:


```
<div id="pic">
<a href="#"><img src="city/0.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/1.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/2.jpg" class="smp" alt="Фото"></a>
<br><a href="#"><img src="city/3.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/4.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/5.jpg" class="smp" alt="Фото"></a>
<br><a href="#"><img src="city/6.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/7.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/8.jpg" class="smp" alt="Фото"></a>
</div>
```

Так как теперь система просмотра изображений стала другой, нам необходимо внести изменения в слои, предназначенные для демонстрации больших

снимков. Если быть точным, фон остается прежним – изменения коснутся содержимого таблицы. Вот что будет в ней:

```
<table id="view">
<tr><td id="td1"> </td>
<td id="td2"><a id="nz" href="#"><img src="img/nz.png"
alt="Назад"></a></td>
<td id="td3"><img id="photo" src="pict/point.jpg" alt="Фото"></td>
<td id="td4"><a id="vp" href="#"><img src="img/vp.png"
alt="Вперед"></a></td>
<td id="td5"><a id="cl" href="#"><img src="img/close.png"
alt="Закрыть"></a></td></tr></table>

#td1 {width: 40px;}
#td2 {text-align: left;}
#td3 {text-align: center;}
#td4 {text-align: right;}
#td5 {text-align: right; vertical-align: top; width: 40px;}
```

Как видите, в таблице появились новые ячейки – их теперь 5. Первая – пустая, она необходима для симметричного размещения всех элементов. Во второй ячейке располагается кнопка обратного просмотра снимков (**id="nz"**). В третьей – само изображение. В четвертой – кнопка прокрутки фотографий вперед (**id="vp"**). В последней ячейке – кнопка завершения просмотра. Как видите, с помощью таблицы мы легко и просто скомпоновали все элементы. Их расположение никак не изменится при просмотре на мониторах с различным разрешением, что нам и надо. Конечно, можно было скомпоновать элементы и другими способами, но использование таблицы заметно упрощает разметку и делает ее более лаконичной. Теперь просмотр изображений выглядит так, как показано на рисунке 1_7.

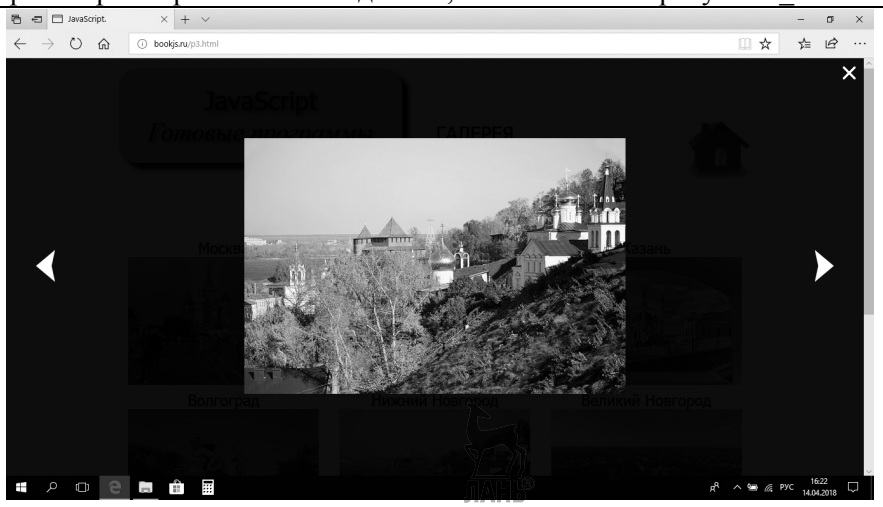

Рис. 1_7. Страница в режиме просмотра галереи. В кадре Нижний Новгород

Напоминаю, что мы удалили возможность завершения просмотра по щелчку на теле таблицы. В противном случае, при нажатии на кнопки прокрутки изображений происходило бы закрытие окна просмотра, так как кнопки – тоже элементы тела таблицы. Но взамен мы добавили отдельную кнопку завершения сеанса:

```
window.onload=function()
{
…
document.getElementById("cl").onclick=del;
};
```

Поэтому у посетителя по-прежнему остались на выбор два варианта возврата на страницу: кликнуть на фото

```
window.onload=function()
{
…
document.getElementById("photo").onclick=del;
…
};
```

или на «крестике». Кроме того, выход из режима просмотра произойдет, если посетитель начнет прокручивать страницу:

```
window.onscroll=del;
```

В самой функции **del()** появились два новых оператора. Они присваивают свойствам **visibility** кнопок прямой и обратной прокрутки значение **hidden.** Если этого не сделать, можно получить неприятный эффект – после того, как вы несколько раз нажмете кнопки и завершите просмотр, фон и фотография исчезнут, а изображения кнопок останутся. Причину возникновения такой ошибки поясним, разбирая механизм работы кнопок.

Функция **del()** теперь выглядит так:

```
function del()
{
document.getElementById("view").style.visibility="hidden";
document.getElementById("bas").style.visibility="hidden";
document.getElementById("vp").style.visibility="hidden";
document.getElementById("nz").style.visibility="hidden";
document.getElementById("photo").src="pict/point.jpg";
return false;
}
```

Глобальные изменения произойдут в функции **look()**:

```
var mas=["0", "1", "2", "3", "4", "5", "6", "7", "8"];

function look(event)
{
if(event.target.tagName=="IMG")
  {
  var sc=window.pageYOffset;
  document.getElementById("bas").style.top=sc+"px";
  document.getElementById("view").style.top=sc+"px";
  document.getElementById("photo").src=event.target.src;
  document.getElementById("vp").style.visibility="visible";
  document.getElementById("nz").style.visibility="visible";
  var posi=document.getElementById("photo").src;
  for(var i=0; i<mas.length; i++)
    {
    if(posi.indexOf("city/"+mas[i]+".jpg")>=0)
      {
      if(i==0)
        {
        document.getElementById("nz").style.visibility="hidden";
        }
      if(i==(mas.length-1))
        {
        document.getElementById("vp").style.visibility="hidden";
        }
      }
    }
  document.getElementById("bas").style.visibility="visible";
  document.getElementById("view").style.visibility="visible";
  }
return false;
}
```

Для начала мы формируем массив **mas** со списком, идентичным именам графических файлов. Файлы у нас называются **0.jpg**, **1.jpg**, **2.jpg** и так далее. В массиве оставлены только их номера без расширения **jpg**. Они необходимы для определения, какой адрес у снимков, стоящих перед демонстрируемым и после него.

Кроме того, в отличие от предыдущих примеров, мы стали учитывать величину прокрутки страницы, которую определяем следующим способом:

```
var sc=window.pageYOffset;
document.getElementById("bas").style.top=sc+"px";
document.getElementById("view").style.top=sc+"px";
```

Поскольку сейчас на нашей странице 9 фотографий, они могут не умещаться целиком в границах экрана, из-за чего браузер создает полосу прокрутки. В первой строке функции узнаем, на сколько пикселей прокручена страница в вертикальном направлении. Во второй и третьей строке смещаем фон и таблицу на вычисленную величину. Имя переменной **sc** образовано от термина screen (экран).

Свойство pageYOffset *содержит информацию о количестве пикселов, на которые страница была прокручена вниз. Для получения информации о величине прокрутки вправо используется свойство* ***pageXOffset****.*

Посетитель может начать просмотр с первого снимка или с любого другого. По клику на маленьком изображении открывается аналогичное фото большого размера и одновременно запускается цикл **for()**, в котором номер открывшегося фото поочередно сравнивается с номерами из массива **mas** и происходит определение, в каком месте галереи находится данный снимок:

```
var posi=document.getElementById("photo").src;
for(var i=0; i<mas.length; i++)
  {
  if(posi.indexOf("city/"+mas[i]+".jpg")>=0)
    {
    …
    }
  }
```

Переменная **posi** (от английского слова position – место, позиция) хранит адрес загруженного снимка. Если он находится в середине списка, то видны обе кнопки прокрутки:

```
document.getElementById("vp").style.visibility="visible";
document.getElementById("nz").style.visibility="visible";
```

Если посетитель запустил просмотр первого фото (**0.jpg**), то кнопка обратной прокрутки не видна:

```
if(i==0)
  {
  document.getElementById("nz").style.visibility="hidden";
  }
```

Когда посетитель смотрит последний снимок (**8.jpg**), то не видна кнопка прокрутки вперед:

```
if(i==(mas.length-1))
  {
  document.getElementById("vp").style.visibility="hidden";
```

```
  }
```

Разберем принцип работы кнопок прокрутки. Начнем с режима просмотра в обратном направлении. Для его реализации у нас есть функция **nz()**:

```
function nz()
{
document.getElementById("vp").style.visibility="visible";
var posi=document.getElementById("photo").src;
for(var i=0; i<mas.length; i++)
  {
  if(posi.indexOf("city/"+mas[i]+".jpg")>=0)
    {
    i--;
    document.getElementById("photo").src="city/"+mas[i]+".jpg";
    if(i==0)
      {
      document.getElementById("nz").style.visibility="hidden";
      }
    break;
    }
  }
return false;
}
```

Метод indexOf(a) *позволяет найти в строке заданный элемент* ***a*** *и возвращает индекс первого найденного элемента. Если же он не обнаружен, возвращается* ***–1****. Поиск ведется от начала строки к концу. Метод чувствителен к регистру.*

Когда посетитель нажимает кнопку прокрутки назад, выполняется первый оператор функции:

```
document.getElementById("vp").style.visibility="visible";
```

Если текущий снимок – последний в списке, то кнопка прокрутки вперед скрыта. В этом случае первая инструкция делает кнопку «Вперед» видимой. Если текущий снимок был не последним, то ничего не изменится: посетитель уже видит кнопку «Вперед».

Дальше в цикле происходит определение номера текущего фото и следующему снимку присваивается адрес, в котором индекс на единицу меньше:

```
i--;
document.getElementById("photo").src="city/"+mas[i]+".jpg";
```

Допустим, вы смотрели фото **5.jpg**. После нажатия кнопки "Назад" атрибуту **src** будет присвоено значение **"city/"+4+".jpg"**, и вы увидите изображение, расположенное по адресу **city/4.jpg**.

Если текущий снимок – первый в списке, то не видна кнопка прокрутки назад:

```
if(i==0)
  {
  document.getElementById("nz").style.visibility="hidden";
  }
```

После того, как в цикле выполнилось условие

if(posi.indexOf("city/"+mas[i]+".jpg")>=0),

и был выполнен один из перечисленных выше операторов, цикл прерывается инструкцией **break**.

Кстати, самое время вспомнить, почему в функции del() мы не ограничились установкой **visibility="hidden"** для всей таблицы, а установили такое значение для каждой кнопки пролистывания в отдельности. Смотрите сами. В процессе пользования галереей мы присваиваем свойству **visibility** одной или обеих кнопок значение **visible**. Но скрытие фона и тела таблицы не меняет значения свойства **visibility** кнопок! Поэтому или одна из них, или обе остаются видимыми, портя общую картину. Так что приходится персонально отключать каждую кнопку прокрутки.

Продолжим. На очереди у нас функция просмотра галереи в прямом направлении:

```
function vp()
{
document.getElementById("nz").style.visibility="visible";
var posi=document.getElementById("photo").src;
for(var i=0; i<mas.length; i++)
  {
  if(posi.indexOf("city/"+mas[i]+".jpg")>=0)
    {
    i++;
    document.getElementById("photo").src="city/"+mas[i]+".jpg";
    if(i==(mas.length-1))
      {
      document.getElementById("vp").style.visibility="hidden";
      }
    break;
    }
  }
return false;
}
```

Она работает по такому же принципу, как и функция **nz()**, только присваивает следующему фото индекс на единицу больше, чем у текущего.

Кнопка «Вперед» делается невидимой по достижении конца списка, а кнопка «Назад» – при достижении начала.

Полный код программы «Галерея» выглядит внушительно:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Галерея</title>
<style>
#bas {position: absolute; z-index: 90; left: 0px; top: 0px; width: 100%; height:
100%; visibility: hidden; background: #000000; opacity: 0.95;}
#view {position: absolute; z-index: 91; left: 0px; top: 0px; width: 100%; height:
100%; visibility: hidden;}
#td1 {width: 40px;}
#td2 {text-align: left;}
#td3 {text-align: center;}
#td4 {text-align: right;}
#td5 {text-align: right; vertical-align: top; width: 40px;}
.smp {width: 300px;}
</style>

<script>
window.onscroll=del;

window.onload=function()
{
document.getElementById("pic").onclick=look;
document.getElementById("nz").onclick=nz;
document.getElementById("vp").onclick=vp;
document.getElementById("photo").onclick=del;
document.getElementById("cl").onclick=del;
};

var mas=["0", "1", "2", "3", "4", "5", "6", "7", "8"];

function look(event)
{
if(event.target.tagName=="IMG")
  {
  var sc=window.pageYOffset;
```

```
  document.getElementById("bas").style.top=sc+"px";
  document.getElementById("view").style.top=sc+"px";
  document.getElementById("photo").src=event.target.src;
  document.getElementById("vp").style.visibility="visible";
  document.getElementById("nz").style.visibility="visible";
  var posi=document.getElementById("photo").src;
  for(var i=0; i<mas.length; i++)
    {
    if(posi.indexOf("city/"+mas[i]+".jpg")>=0)
      {
      if(i==0)
        {
        document.getElementById("nz").style.visibility="hidden";
        }
      if(i==(mas.length-1))
        {
        document.getElementById("vp").style.visibility="hidden";
        }
      }
    }
  document.getElementById("bas").style.visibility="visible";
  document.getElementById("view").style.visibility="visible";
  }
return false;
}

function nz()
{
document.getElementById("vp").style.visibility="visible";
var posi=document.getElementById("photo").src;
for(var i=0; i<mas.length; i++)
  {
  if(posi.indexOf("city/"+mas[i]+".jpg")>=0)
    {
    i--;
    document.getElementById("photo").src="city/"+mas[i]+".jpg";
    if(i==0)
      {
      document.getElementById("nz").style.visibility="hidden";
      }
    break;
    }
  }
return false;
}
```

```
function vp()
{
document.getElementById("nz").style.visibility="visible";
var posi=document.getElementById("photo").src;
for(var i=0; i<mas.length; i++)
  {
  if(posi.indexOf("city/"+mas[i]+".jpg")>=0)
    {
    i++;
    document.getElementById("photo").src="city/"+mas[i]+".jpg";
    if(i==(mas.length-1))
      {
      document.getElementById("vp").style.visibility="hidden";
      }
    break;
    }
  }
return false;
}

function del()
{
document.getElementById("view").style.visibility="hidden";
document.getElementById("bas").style.visibility="hidden";
document.getElementById("vp").style.visibility="hidden";
document.getElementById("nz").style.visibility="hidden";
document.getElementById("photo").src="pict/point.jpg";
return false;
}
</script>
</head>

<body>

<div id="bas"></div>
<table id="view">
<tr><td id="td1"> </td>
<td id="td2"><a id="nz" href="#"><img src="img/nz.png"
alt="Назад"></a></td>
<td id="td3"><img id="photo" src="pict/point.jpg" alt="Фото"></td>
<td id="td4"><a id="vp" href="#"><img src="img/vp.png"
alt="Вперед"></a></td>
<td id="td5"><a id="cl" href="#"><img src="img/close.png"
alt="Закрыть"></a></td></tr></table>
```

```
<div id="pic">
<a href="#"><img src="city/0.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/1.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/2.jpg" class="smp" alt="Фото"></a>
<br><a href="#"><img src="city/3.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/4.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/5.jpg" class="smp" alt="Фото"></a>
<br><a href="#"><img src="city/6.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/7.jpg" class="smp" alt="Фото"></a>
<a href="#"><img src="city/8.jpg" class="smp" alt="Фото"></a>
</div>

</body>
</html>
```

Впрочем, пусть вас не пугает большой объем кода: это один из самых сложных примеров в данной книге. Большинство будет все-таки попроще. К тому же вы имеете возможность скопировать данный код с сайта поддержки: http://bookjs.ru/code/p3.txt.

Адаптируя программу для своих проектов, помните, что вам надо будет внести следующие изменения:

– увеличить или уменьшить количество элементов списка **mas** в зависимости от количества фотографий на вашей странице;

– внести соответствующие изменения в адреса фото и в описания функций, если название вашей папки с изображениями будет отличаться от **city**.

Вот и все. В следующем разделе мы поговорим о других разновидностях галерей.

Слайд-шоу

«Фишка» многих ресурсов – дать посетителям возможность выбирать разные способы просмотра галерей: в ручном режиме или в автоматическом. Последний еще принято называть слайд-шоу. У всех разные вкусы и способность программиста удовлетворить их повышает рейтинг сайта. Ну что ж, мы не станем отставать от лидеров web-индустрии и создадим свое слайд-шоу. Более того, у нас будет два варианта автоматической демонстрации снимков.

Сразу предупрежу: оба варианта мы будем рассматривать по отдельности. В наши планы не входит соединять оба примера в один или добавлять к ним «ручную» галерею. Иначе код такой версии получится слишком громоздким для этой книги. Читателям, решившим сделать «навороченную» галерею, придется сводить три разных кода в одну страницу самостоятельно.

Хотя оба варианта демонстрируются на одной и той же странице – http://bookjs.ru/p4.html – они, как видите, имеют разные ссылки на примеры кода (рис. 1_8). Для запуска слайд-шоу есть две кнопки: «Пример 1» и «При-

мер 2». Чтобы вернуть снимки в начальное состояние, надо воспользоваться третьей кнопкой «В ИСХОДНОЕ».

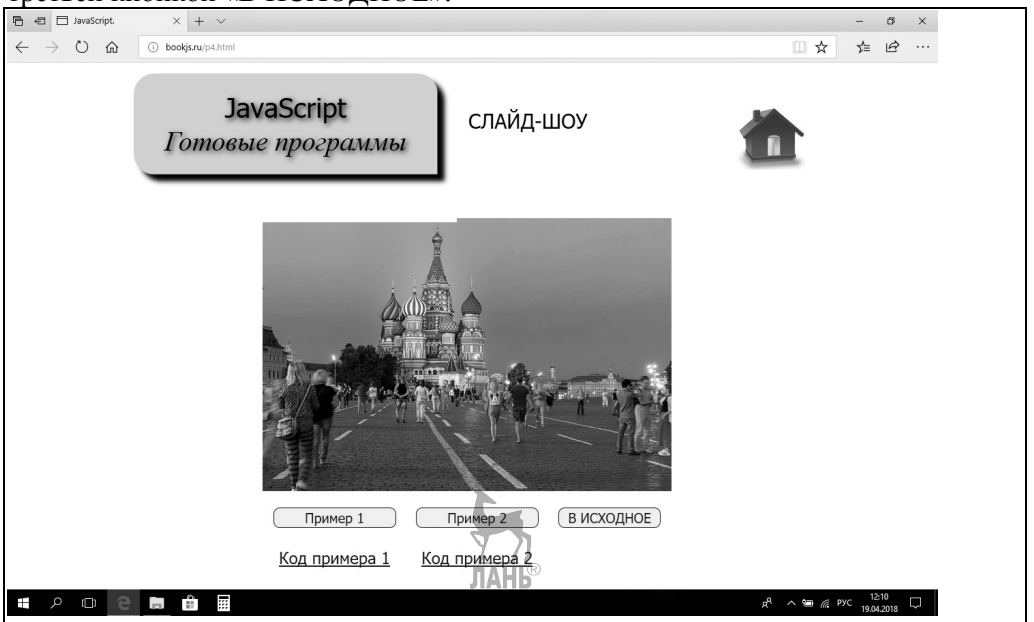

Рис. 1_8. Страница с примерами слайд-шоу. Просмотр фотографий начинается со столицы

Для обоих примеров необходимо сначала изменить расположение фотографий. Если раньше они стояли последовательно одна за другой, то теперь мы сложим их «стопкой». На самом верху будет снимок Москвы, ниже – Санкт-Петербурга, потом Казани – и далее по списку. На этот раз все изображения сразу будут естественного размера:

```
<div>
<img src="city/0.jpg" id="p0" alt="Фото">
<img src="city/1.jpg" id="p1" alt="Фото">
<img src="city/2.jpg" id="p2" alt="Фото">
<img src="city/3.jpg" id="p3" alt="Фото">
<img src="city/4.jpg" id="p4" alt="Фото">
<img src="city/5.jpg" id="p5" alt="Фото">
<img src="city/6.jpg" id="p6" alt="Фото">
<img src="city/7.jpg" id="p7" alt="Фото">
<img src="city/8.jpg" id="p8" alt="Фото">
</div>

div {position: relative; z-index: 1; width: 1000px; margin: 0px auto;}
#p0 {position: absolute; z-index: 15; left: 200px; top: 20px;}
#p1 {position: absolute; z-index: 14; left: 200px; top: 20px;}
#p2 {position: absolute; z-index: 13; left: 200px; top: 20px;}
#p3 {position: absolute; z-index: 12; left: 200px; top: 20px;}
```

```
#p4 {position: absolute; z-index: 11; left: 200px; top: 20px;}
#p5 {position: absolute; z-index: 10; left: 200px; top: 20px;}
#p6 {position: absolute; z-index: 9; left: 200px; top: 20px;}
#p7 {position: absolute; z-index: 8; left: 200px; top: 20px;}
#p8 {position: absolute; z-index: 7; left: 200px; top: 20px;}
```

«Пачку» фотографий мы поместили в центре страницы. Для первого примера место, где находятся снимки, не играет роли, а для второго оно принципиально – механизм работы в этом случае построен именно на центральном положении изображений.

Важное отличие, которое есть между демонстрационной страницей и кодом примеров: на странице просмотр запускается нажатием кнопки, в примерах код упрощен и запуск слайд-шоу происходит автоматически сразу после загрузки страницы. Так сделано, чтобы избавиться от лишних кнопок и максимально сократить разметку. Естественно, что, внедряя код в свой сайт, вы можете прибегнуть к любому методу включения слайд-шоу.

Как работает первый вариант? Нажмите кнопку «Пример 1» – и вы увидите, как спустя пару секунд изображение Москвы медленно «растает», а сквозь него «проявится» фото Санкт-Петербурга (рис. 1_9). Затем так же плавно исчезнет снимок города на Неве и появится Казань. И так далее до конца галереи. Последним идет Псков. Когда его изображение «растворится», вы вновь увидите Москву. Далее цикл повторится – и так до бесконечности (точнее, до тех пор, пока открыта страница или до тех пор, пока вы не нажмете кнопку «В ИСХОДНОЕ»). Если код будет использован в вашем проекте, вы можете предусмотреть любой подходящий способ прерывания цикла.

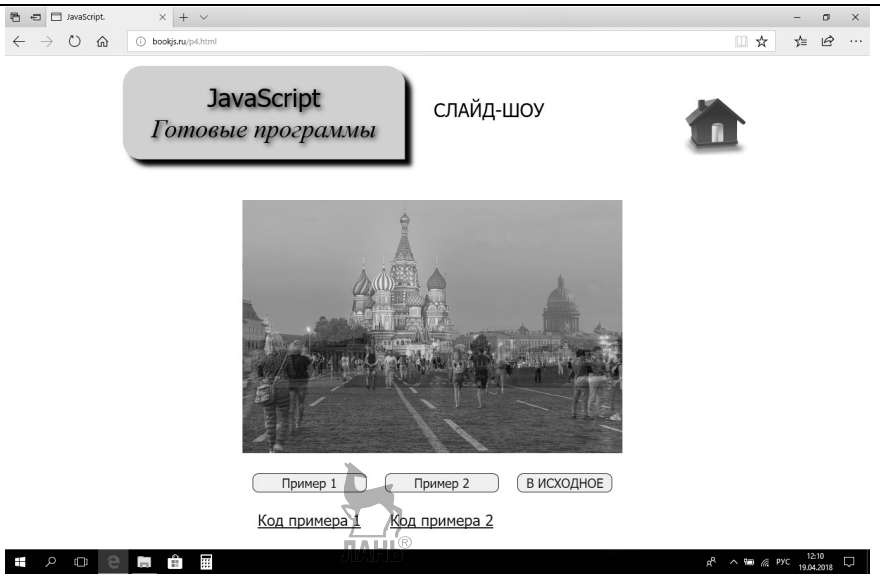

Рис. 1_9. Первый вариант слайд-шоу в действии: проявление нижнего изображения сквозь верхнее

Для автоматического запуска слайд-шоу зарегистрируем обработчик события:

```
window.onload=cycle;
```

Имя **cycle** на мой взгляд отлично походит к данной функции. Ее код:

```
var t=2;
var d=0;

function cycle()
{
var ph="p"+d;
if(d<8)
  {
  if(t<=0)
    {
    d++;
    t=2;
    cycle();
    }
  else
    {
    document.getElementById(ph).style.opacity=t;
    t=t-0.01;
    window.setTimeout(cycle, 20);
    }
  }
else
  {
  document.getElementById("p0").style.zIndex=6;
  document.getElementById("p0").style.opacity=2;
  if(t<=0)
    {
    document.getElementById("p0").style.zIndex=15;
    document.getElementById("p0").style.opacity=2;
    document.getElementById("p1").style.opacity=2;
    document.getElementById("p2").style.opacity=2;
    document.getElementById("p3").style.opacity=2;
    document.getElementById("p4").style.opacity=2;
    document.getElementById("p5").style.opacity=2;
    document.getElementById("p6").style.opacity=2;
    document.getElementById("p7").style.opacity=2;
    document.getElementById("p8").style.opacity=2;
    d=0;
```

```
    ph="p0";
    t=2;
    cycle();
    }
  else
    {
    document.getElementById(ph).style.opacity=t;
    t=t-0.01;
    window.setTimeout(cycle, 20);
    }
  }
}
```

Переменная **t** является счетчиком уровня прозрачности **opacity**. Начальное значение **t** – число 2. Переменная **d** – счетчик снимков. В переменную **ph** поочередно «загружается» ID каждого снимка (**ph="p"+d**).

Внимательный читатель удивится: почему **t=2**, если максимальный показатель прозрачности не может превышать единицу? При любом значении **opacity** больше единицы слой будет полностью непрозрачным. Тогда зачем нужна эта двойка? Логика выбора начального значения **t** станет ясна в процессе разбора функции **cycle()**.

После запуска слайд-шоу сразу начинает выполняться блок инструкций

```
document.getElementById(ph).style.opacity=t;
t=t-0.01;
window.setTimeout(cycle, 20);
```

Метод setTimeout(f, t) *по сути является таймером, однократно вызывающим некоторую функцию (например, созданную разработчиком) или однократно выполняющим какую-либо инструкцию через определенный интервал времени. Метод оперирует двумя аргументами:* ***f*** *– имя вызываемой функции или инструкция,* ***t*** *– временной интервал в миллисекундах. Метод может вызываться:*
– так ***window.setTimeout(func, 1000);***
– так ***window.setTimeout('document.getElementById("pict").style.visibility="visible"', 1000);***
– и даже так ***window.setTimeout('document.getElementById("pict").style.visibility="visible"; func()', 1000)***

Свойству **opacity** первого снимка присваивается текущее значение переменной **t** – число 2. И тут же **t** уменьшается на 0.01 (запись **t=t-0.01** я использовал, чтобы неискушенному читателю было проще понять, что делается в этой строке; более опытный программист может написать короче: **t-=0.01**). Через 20 миллисекунд происходит новый вызов функции **cycle()**. **Opacity** уже 1.99, из **t** вновь вычитается 0.01, спустя 20 миллисекунд повторяется вызов функции. Как видите, в течении первых двух секунд состояние снимка **0.jpg** не меняется. Этого времени достаточно, чтобы посетитель рассмотрел изображение – вот в чем «сакральный» смысл числа 2. Затем уровень прозрачности преодолевает

рубеж «1» и фотография начинает плавно «растворяться». Когда **opacity** достигает нулевой отметки, процесс останавливается и выполняется блок операторов

```
if(t<=0)
  {
  d++;
  t=2;
  cycle();
  }
```

Переменной **d** присваивается значение 1, поэтому следующие операции производятся над фото **1.jpg** (оно второе в нашей «пачке»). Переменной **t** снова присваивается значение 2. Фотография Санкт-Петербурга будет две секунды доступна для просмотра, а затем уступит место Казани. И так далее. Наконец появится изображение Пскова. Переменная **d** примет значение 8 и выполнятся следующие инструкции:

```
document.getElementById("p0").style.zIndex=6;
document.getElementById("p0").style.opacity=2;
```

В результате изображение Москвы с вершины нашей фотографической пирамиды переместится в ее основание и вновь станет полностью непрозрачным. За счет этого Псков будет «растворятся» не на белом фоне страницы, а на фоне столицы. Когда снимок Пскова исчезнет, произойдет переустановка свойств всех картинок в исходное состояние:

```
document.getElementById("p0").style.zIndex=15;
document.getElementById("p0").style.opacity=2;
document.getElementById("p1").style.opacity=2;
document.getElementById("p2").style.opacity=2;
document.getElementById("p3").style.opacity=2;
document.getElementById("p4").style.opacity=2;
document.getElementById("p5").style.opacity=2;
document.getElementById("p6").style.opacity=2;
document.getElementById("p7").style.opacity=2;
document.getElementById("p8").style.opacity=2;
```

Этот фрагмент можно было бы упростить, выполнив операцию **opacity=2** для каждого фото в цикле for или while. Но я решил оставить более длинную запись для наглядности. При желании читателю не составит труда внести необходимые изменения.

Последние четыре оператора

```
d=0;
ph="p0";
```

```
t=2;
cycle();
```

установят счетчикам начальные значения и запустят очередной виток цикла.

Код полностью:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Слайд-шоу 1</title>
<style>
div {position: relative; z-index: 1; width: 1000px; margin: 0px auto;}
#p0 {position: absolute; z-index: 15; left: 200px; top: 20px;}
#p1 {position: absolute; z-index: 14; left: 200px; top: 20px;}
#p2 {position: absolute; z-index: 13; left: 200px; top: 20px;}
#p3 {position: absolute; z-index: 12; left: 200px; top: 20px;}
#p4 {position: absolute; z-index: 11; left: 200px; top: 20px;}
#p5 {position: absolute; z-index: 10; left: 200px; top: 20px;}
#p6 {position: absolute; z-index: 9; left: 200px; top: 20px;}
#p7 {position: absolute; z-index: 8; left: 200px; top: 20px;}
#p8 {position: absolute; z-index: 7; left: 200px; top: 20px;}
</style>

<script>
window.onload=cycle;

var t=2;
var d=0;

function cycle()
{
var ph="p"+d;
if(d<8)
  {
  if(t<=0)
    {
    d++;
    t=2;
    cycle();
    }
  else
    {
    document.getElementById(ph).style.opacity=t;
    t=t-0.01;
```

```
    window.setTimeout(cycle, 20);
    }
  }
else
  {
  document.getElementById("p0").style.zIndex=6;
  document.getElementById("p0").style.opacity=2;
  if(t<=0)
    {
    document.getElementById("p0").style.zIndex=15;
    document.getElementById("p0").style.opacity=2;
    document.getElementById("p1").style.opacity=2;
    document.getElementById("p2").style.opacity=2;
    document.getElementById("p3").style.opacity=2;
    document.getElementById("p4").style.opacity=2;
    document.getElementById("p5").style.opacity=2;
    document.getElementById("p6").style.opacity=2;
    document.getElementById("p7").style.opacity=2;
    document.getElementById("p8").style.opacity=2;
    d=0;
    ph="p0";
    t=2;
    cycle();
    }
  else
    {
    document.getElementById(ph).style.opacity=t;
    t=t-0.01;
    window.setTimeout(cycle, 20);
    }
  }
}
</script>
</head>

<body>

<div>
<img src="city/0.jpg" id="p0" alt="Фото">
<img src="city/1.jpg" id="p1" alt="Фото">
<img src="city/2.jpg" id="p2" alt="Фото">
<img src="city/3.jpg" id="p3" alt="Фото">
<img src="city/4.jpg" id="p4" alt="Фото">
<img src="city/5.jpg" id="p5" alt="Фото">
<img src="city/6.jpg" id="p6" alt="Фото">
```

```
<img src="city/7.jpg" id="p7" alt="Фото">
<img src="city/8.jpg" id="p8" alt="Фото">
</div>

</body>
</html>
```

Посмотрим, как работает другой вариант слайд-шоу. Для этого нажмем кнопку «В ИСХОДНОЕ», а затем «Пример 2». Фотографии расположатся в первоначальном порядке, а по истечении двух секунд верхний снимок плавно «уедет» влево за границы экрана. Как это происходит, видно на рисунке 1_10. Спустя небольшой промежуток времени за первой фотографией последует вторая, за ней третья – и так до последнего снимка. Интервал между исчезновением одной фотографии и началом движения другой зависит от размеров монитора вашего компьютера. В отличие от предыдущего варианта слайд-шоу, в этом процесс не зациклен, а значит, отправив все снимки за границы экрана, программа оставит на месте изображений пустой фон.

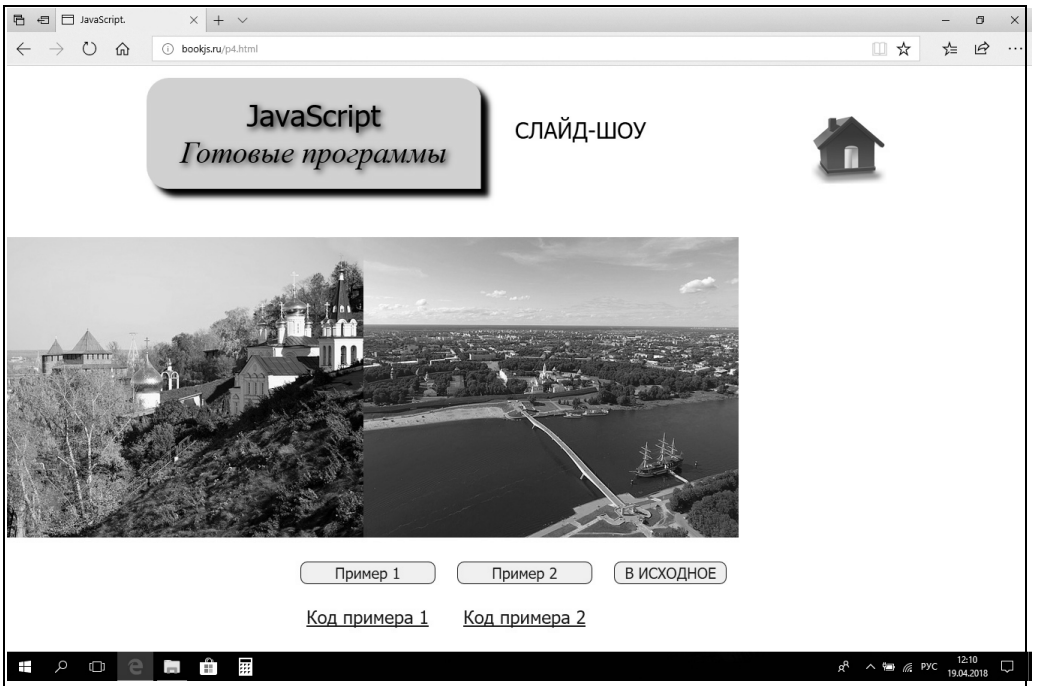

Рис. 1_10. Второй вариант слайд-шоу. В кадр попал момент, когда фотография Нижнего Новгорода сменяется снимком Великого Новгорода

Как и в предыдущем случае, предложу вашему вниманию код, в котором запуск слайд-шоу происходит автоматически после загрузки страницы:

```
window.onload=drive;
```

Перед функцией **drive()** (в переводе на русский язык – ездить) объявляем три переменные:

```
var i=1;
var k=200;
var s=0;
```

Переменная **i** – идентификатор начала процесса. Рассматривая работу программы, мы первым делом объясним ее назначение. **k** – счетчик смещения фотографии влево, **s** – счетчик снимков.

Теперь сама функция:

```
function drive()
{
var ph="p"+s;
if(i==1)
  {
  i=0;
  window.setTimeout(drive, 2000);
  }
else
  {
  if(s<9)
    {
    if(k>=-3000)
      {
      document.getElementById(ph).style.left=k+"px";
      k=k-10;
      window.setTimeout(drive, 10);
      }
    else
      {
      k=200;
      s++;
      window.setTimeout(drive, 10);
      }
    }
  }
}
```

Как и в предыдущем случае, в переменную **ph** поочередно «загружается» ID каждого снимка (**ph="p"+s**).

Первым выполняется блок

```
if(i==1)
  {
  i=0;
  window.setTimeout(drive, 2000);
  }
```

Если бы он отсутствовал, верхнее изображение начало бы движение сразу после загрузки страницы. При наличии такого блока мы даем посетителю 2 секунды на просмотр фотографии Москвы. Затем **i** становится равно нулю и функция вызывается заново. Теперь первый блок пропускается, так как условие **if(i==1)** ложно, и начинают выполняться следующие инструкции. Таким образом, программа, «ориентируясь» на значение переменной **i**, «понимает», первый это вызов или нет, надо задержать верхний снимок или нет.

Дальше все просто. Свойству **left** фотографии с **id=ph** присваивается значение переменной **k**. Сейчас у нас идет речь о верхнем снимке, поэтому **id="p0"**, а значение **left** составляет **200px** (начальное значение **k=200** выбрано именно таким, поскольку в исходном положении все фотографии внутри слоя **div** смещены на 200 пикселей). Происходит выполнение оператора **k=k-10** и функция **drive()** спустя 10 миллисекунд вызывается снова (как и в предыдущем примере, запись **k=k-10** можно укоротить: **k-=10**). Теперь значение **left** составляет **190px** и посетитель видит, как изображение начинает смещаться влево. Так повторяется до тех пор, пока выполняется условие **k>=-3000**. То есть снимок «уходит» влево в общей сложности на 3200 пикселей от начального положения. Этого достаточно, чтобы скрыться с экрана самого большого монитора.

Когда верхняя фотография полностью исчезла, программа переходит к следующему блоку инструкций:

```
k=200;
s++;
window.setTimeout(drive, 10);
```

Теперь переменная **s** увеличивается на единицу и функция приступает к работе с фотоснимком **1.jpg**. Поскольку условие **k>=-3000** снова истинно, повторяется выполнение блока

```
document.getElementById(ph).style.left=k+"px";
k=k-10;
window.setTimeout(drive, 10);
```

Затем исчезает фото **1.jpg** и программа «берется» за третий снимок. Таким образом, дело до ходит до последнего – **8.jpg**. После его исчезновения условие **if(s<9)** становится ложным и выполнение программы прерывается. Просмотр слайд-шоу завершен.

Соберем страницу в единое целое:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Слайд-шоу 2</title>
<style>
div {position: relative; z-index: 1; width: 1000px; margin: 0px auto;}
#p0 {position: absolute; z-index: 15; left: 200px; top: 20px;}
#p1 {position: absolute; z-index: 14; left: 200px; top: 20px;}
#p2 {position: absolute; z-index: 13; left: 200px; top: 20px;}
#p3 {position: absolute; z-index: 12; left: 200px; top: 20px;}
#p4 {position: absolute; z-index: 11; left: 200px; top: 20px;}
#p5 {position: absolute; z-index: 10; left: 200px; top: 20px;}
#p6 {position: absolute; z-index: 9; left: 200px; top: 20px;}
#p7 {position: absolute; z-index: 8; left: 200px; top: 20px;}
#p8 {position: absolute; z-index: 7; left: 200px; top: 20px;}
</style>
<script>
window.onload=drive;

var i=1;
var k=200;
var s=0;

function drive()
{
var ph="p"+s;
if(i==1)
  {
  i=0;
  window.setTimeout(drive, 2000);
  }
else
  {
  if(s<9)
    {
    if(k>=-3000)
      {
      document.getElementById(ph).style.left=k+"px";
      k=k-10;
      window.setTimeout(drive, 10);
      }
    else
      {
      k=200;
```

```
        s++;
        window.setTimeout(drive, 10);
        }
      }
    }
}
</script>
</head>

<body>

<div>
<img src="city/0.jpg" id="p0" alt="Фото">
<img src="city/1.jpg" id="p1" alt="Фото">
<img src="city/2.jpg" id="p2" alt="Фото">
<img src="city/3.jpg" id="p3" alt="Фото">
<img src="city/4.jpg" id="p4" alt="Фото">
<img src="city/5.jpg" id="p5" alt="Фото">
<img src="city/6.jpg" id="p6" alt="Фото">
<img src="city/7.jpg" id="p7" alt="Фото">
<img src="city/8.jpg" id="p8" alt="Фото">
</div>

</body>
</html>
```

Если вы захотите сделать второй пример цикличным, воспользуйтесь подходом из первого примера. Когда последняя фотография скроется за рамками экрана, необходимо присвоить всем переменным и свойствам **left** всех снимков их начальное значение, после чего вновь вызвать функцию **drive()**.

Загрузка по частям

Во всех примерах, рассмотренных ранее, мы имели дело с небольшим количеством фотографий. Когда снимков немного, их можно разместить на странице все сразу. Сложнее обстоит дело, если фотографий, например, несколько десятков. Имеет ли смысл показывать всю галерею целиком? Или лучше дать возможность посетителю самостоятельно решить, сколько снимков он хочет просмотреть: все или только часть из них?

На многих сайтах используется подход, при котором информация выдается по частям, определенными порциями, а посетитель сам решает, когда ему остановить просмотр. Пример – социальные сети. В них сообщения и фотографии в ленту событий загружаются по несколько штук по мере прокрутки страницы в окне браузера.

У нас уже складывается традиция ориентироваться на передовые методы программирования. Не будем делать исключений и в данной ситуации. Попробуем написать код, благодаря которому наши фото станут отображаться на странице по частям. Только сразу оговорим, что в данном случае наш способ не будет точной копией того, что применяется в социальных сетях. У нас загрузка очередных изображений станет происходить не в результате прокрутки страницы, а при нажатии на кнопку или на ссылку «ЕЩЕ ФОТО...».

Что касается технологий, используемых в социальных сетях, то к ним мы еще вернемся, когда в пятой главе рассмотрим создание бесконечной ленты новостей.

Итак, приступим.

Как всегда, начнем с того, что создадим простую HTML-страницу с девятью фотографиями городов. Только теперь три из них видны сразу, а остальные до поры до времени скрыты от посетителя (рис. 1_11).

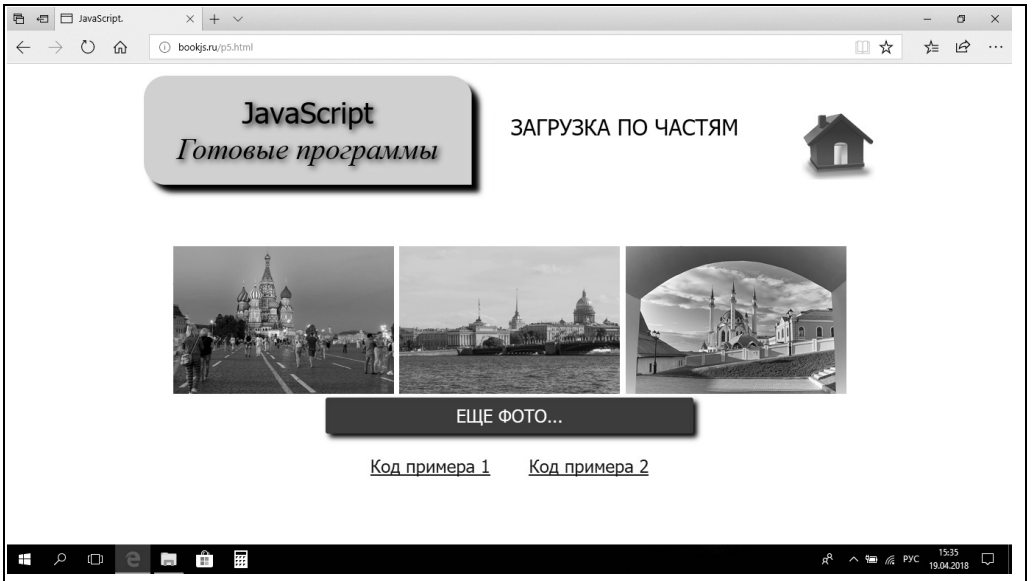

Рис. 1_11. Страница, иллюстрирующая метод загрузки фотографий по частям

Фотографии в теле страницы разместятся таким образом:

```
<img src="city/0.jpg" alt="Фото">
<img src="city/1.jpg" alt="Фото">
<img src="city/2.jpg" alt="Фото"><br>
<a id="dis1" href="#">ЕЩЕ ФОТО...</a>

<div id="ph1" class="phot">
<img src="city/3.jpg" alt="Фото">
<img src="city/4.jpg" alt="Фото">
<img src="city/5.jpg" alt="Фото"><br>
<a id="dis2" href="#">ЕЩЕ ФОТО...</a>
</div>
```

```
<div id="ph2" class="phot">
<img src="city/6.jpg" alt="Фото">
<img src="city/7.jpg" alt="Фото">
<img src="city/8.jpg" alt="Фото">
</div>
```

Свойства фотографий и слоев:

```
img {width: 300px;}
.phot {display: none;}
```

Когда страница загружается, посетитель видит изображения Москвы, Санкт-Петербурга и Казани. Остальные снимки скрыты в слоях **div**, свойствам **display** которых присвоено значение **none**. В нашем случае такой подход очень удобен: на начальном этапе последние 6 фотографий не просто не видны, а вообще не занимают никакого места на странице. Для просмотра очередной порции снимков существуют две ссылки:

```
<a id="dis1" href="#">ЕЩЕ ФОТО...</a>
<a id="dis2" href="#">ЕЩЕ ФОТО...</a>
```

В начальный момент посетитель видит только первую, вторая скрыта вместе с остальными фотографиями.

Функция **newph()** (от английского new photo), запускающая просмотр очередной группы снимков, очень простая:

```
var a=0;

function newph()
{
a++;
var nam_di="dis"+a;
var nam_ph="ph"+a;
document.getElementById(nam_di).style.display="none";
document.getElementById(nam_ph).style.display="inline";
var b="dis"+(a+1);
if(document.getElementById(b))
  {
  document.getElementById(b).onclick=newph;
  }
return false;
}
```

Сначала создаем переменную **a**. Она выполняет роль счетчика, указывающего, какой слой необходимо сделать видимым, а какую кнопку или ссылку нужно, наоборот, скрыть.

В исходном состоянии, как мы уже выяснили, видна только первая ссылка. Она вызывает функцию **newph()**. По щелчку на ссылке счетчик получит значение 1, а переменным **nam_di** и **nam_ph** будут присвоены значения **dis1** и **ph1** соответственно. В результате первая ссылка с **id="dis1"** скроется, а слой с **id="ph1"**, наоборот, станет видимым. Появятся фотографии Волгограда, Нижнего Новгорода и Великого Новгорода, а под ними – вторая ссылка (рис. 1_12). Нажатие на эту ссылку, по аналогии с предыдущим случаем, приведет к ее скрытию и появлению еще трех фотографий. Как видите, все очень просто. Пример в действии и его код можно увидеть по адресу http://bookjs.ru/p5.html.

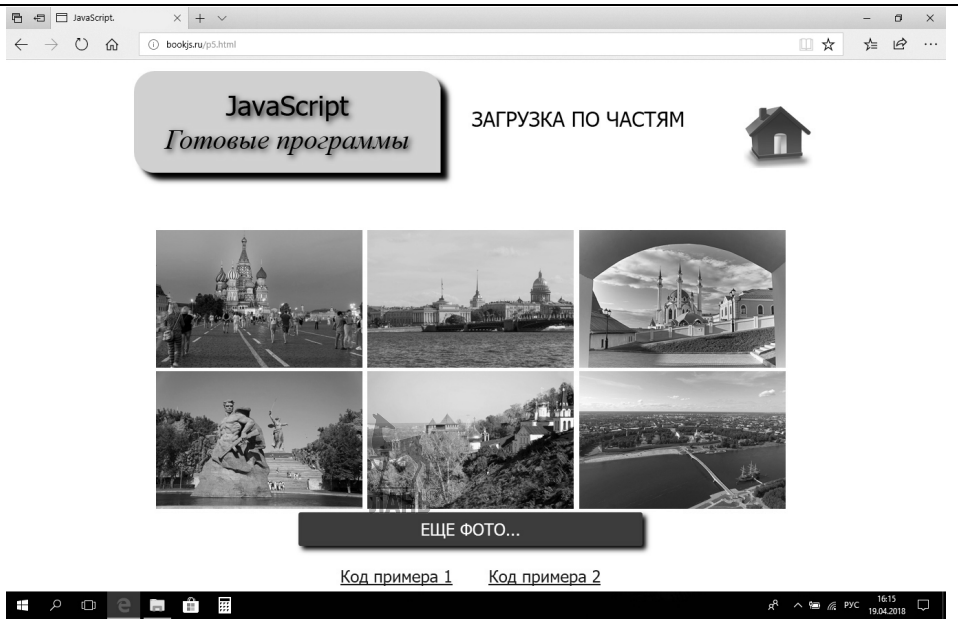

Рис. 1_12. После нажатия кнопки «ЕЩЕ ФОТО...» на странице появляется очередная группа фотографий

Чтобы наша функция работала, зарегистрируем обработчик события click для первой ссылки:

```
window.onload=function()
{
document.getElementById("dis1").onclick=newph;
};
```

В теле функции есть интересный фрагмент:

```
var b="dis"+(a+1);
if(document.getElementById(b))
  {
  document.getElementById(b).onclick=newph;
  }
```

В этой части программы создается обработчик события для второй ссылки. В принципе, это можно было сделать одновременно с регистрацией обработчика для первой. Но такой подход таит в себе одно существенное неудобство. Если бы мы захотели увеличить количество слоев с изображениями, нам пришлось бы добавлять в блок

```
window.onload=function()
{
…
};
```

столько же новых инструкций, сколько мы добавили новых слоев. Мы поступили хитрее и с помощью последних строк в функции **newph()** просто автоматизировали этот процесс.

Когда все фото откроются, нужно остановить создание очередного обработчика для группы снимков с **id** на единицу больше последнего – ведь такого слоя нет. Поэтому мы сначала проверяем наличие следующего слоя

```
if(document.getElementById(b))
```

и только потом инициируем следующий обработчик.

В «собранном» виде наша страница выглядит так:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Загрузка по частям</title>
<style>
img {width: 300px;}
.phot {display: none;}
</style>

<script>
window.onload=function()
{
document.getElementById("dis1").onclick=newph;
};

var a=0;
```

```
function newph()
{
a++;
var nam_di="dis"+a;
var nam_ph="ph"+a;
document.getElementById(nam_di).style.display="none";
document.getElementById(nam_ph).style.display="inline";
var b="dis"+(a+1);
if(document.getElementById(b))
  {
  document.getElementById(b).onclick=newph;
  }
return false;
}
</script>
</head>

<body>

<img src="city/0.jpg" alt="Фото">
<img src="city/1.jpg" alt="Фото">
<img src="city/2.jpg" alt="Фото"><br>
<a id="dis1" href="#">ЕЩЕ ФОТО...</a>

<div id="ph1" class="phot">
<img src="city/3.jpg" alt="Фото">
<img src="city/4.jpg" alt="Фото">
<img src="city/5.jpg" alt="Фото"><br>
<a id="dis2" href="#">ЕЩЕ ФОТО...</a>
</div>

<div id="ph2" class="phot">
<img src="city/6.jpg" alt="Фото">
<img src="city/7.jpg" alt="Фото">
<img src="city/8.jpg" alt="Фото">
</div>

</body>
</html>
```

К сожалению, в этой простоте кроется одна серьезная неприятность. При **display: none** скрытые фотографии не видны на странице, но они загружаются в память браузера. Когда у нас девять снимков, это не критично. А если фотографий несколько десятков, то получится так: вроде бы на начальном этапе снимков немного, но страница почему-то долго загружается. А все дело в десятках

невидимых изображений, которые грузятся одновременно с видимым контентом.

Обойти эту неприятность позволит второй вариант отложенной загрузки.

Наша задача сделать так, чтобы код скрытых изображений хранился в памяти браузера в виде простого текста. Для решения этой задачи мы прибегнем к помощи поля формы **textarea**. Если вы уже использовали **textarea** в своих проектах, то прекрасно знаете, что любой код, помещенный внутри открывающего и закрывающего тегов, выглядит как простой текст и не преобразуется в соответствующие элементы страницы. Разместив наши изображения в полях **textarea**, мы избежим преждевременной загрузки фото и страница станет открываться гораздо быстрее.

Содержимое тега **body** примет следующий вид:

```
<div id="gal">
<img src="city/0.jpg" alt="Фото">
<img src="city/1.jpg" alt="Фото">
<img src="city/2.jpg" alt="Фото"><br>
<a id="dis1" href="#">ЕЩЕ ФОТО...</a>
</div>

<textarea id="ph1" class="phot">
<img src="city/3.jpg" alt="Фото">
<img src="city/4.jpg" alt="Фото">
<img src="city/5.jpg" alt="Фото"><br>
<a id="dis2" href="#">ЕЩЕ ФОТО...</a>
</textarea>

<textarea id="ph2" class="phot">
<img src="city/6.jpg" alt="Фото">
<img src="city/7.jpg" alt="Фото">
<img src="city/8.jpg" alt="Фото">
</textarea>
```

Настройки стилей будут выглядеть так:

```
img {width: 300px;}
.phot {position: fixed; left: 0px; top: 0px; visibility: hidden;}
```

Теперь изначально открытые снимки разместились в слое ** ** (от английского gallery). Сюда же мы будем постепенно добавлять и остальные фото.

У нас появились 2 поля **textarea**, расположенные в левом верхнем углу и не привязанные к остальной разметке. Поля скрыты от посетителя и не мешают просмотру страницы. В каждом из них по 3 фотографии. Кроме того, в первом поле есть ссылка для запуска функции **newph()**.

Функция **newph()** для второго примера выглядит так:

```
var a=0;

function newph()
{
a++;
var nam_di="dis"+a;
var nam_ph="ph"+a;
document.getElementById(nam_di).style.display="none";
var con=document.getElementById(nam_ph).value;
document.getElementById("gal").insertAdjacentHTML("beforeend", con);
var b="dis"+(a+1);
if(document.getElementById(b))
  {
  document.getElementById(b).onclick=newph;
  }
return false;
}
```

Щелчок на первой ссылке запускает функцию и переменная **a** принимает значение 1. Первая ссылка скрывается от клиента. Одновременно переменной **con** (от английского content) присваивается содержимое **textarea** с **id="ph1"**. К уже существующему содержанию страницы с помощью метода **insertAdjacentHTML()** добавляются новые фотоснимки:

```
document.getElementById("gal").insertAdjacentHTML("beforeend", con);
```

Значение **beforeend** в первом аргументе указывает, что новое содержимое необходимо добавить в конец контейнера **gal** перед закрывающим тегом ** **. Аналогичные события происходят по щелчку на второй ссылке – она скрывается, а на страницу загружаются еще 3 фотографии.

Метод insertAdjacentHTML() *предназначен для добавления данных (они могут быть в виде HTML-кода или простого текста) к какому-либо элементу страницы. Первый аргумент метода указывает на место вставки, второй содержит код или текст. Если элемент имеет открывающий и закрывающий теги, то данные могут быть добавлены непосредственно перед открывающим тегом (**beforebegin**), сразу после него (**afterbegin**), непосредственно перед закрывающим тегом (**beforeend**) и сразу после него (**afterend**). Если элемент состоит из единственного тега, то данные могут быть добавлены непосредственно перед ним или сразу после него.*

В новом варианте инструкции

```
var b="dis"+(a+1);
if(document.getElementById(b))
```

```
{
document.getElementById(b).onclick=newph;
}
```

имеют двойное значение. Про первое мы уже говорили – это способ автоматизировать создание новых обработчиков событий. Чтобы понять второе, вспомним, что скрытые фотографии и ссылка хранятся в **textarea** в виде простого текста. Поэтому создать обработчик события click для второй ссылки в блоке

```
window.onload=function()
{
…
};
```

не удастся. Ведь в момент загрузки страницы вторая ссылка еще не существует! Следовательно, единственная возможность инициировать обработчик – только внутри функции после записи содержимого **textarea** с **id="ph1"** в слой **gal**.

Полный код второго варианта:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Загрузка по частям</title>
<style>
img {width: 300px;}
.phot {position: fixed; left: 0px; top: 0px; visibility: hidden;}
</style>

<script>
window.onload=function()
{
document.getElementById("dis1").onclick=newph;
};

var a=0;

function newph()
{
a++;
var nam_di="dis"+a;
var nam_ph="ph"+a;
document.getElementById(nam_di).style.display="none";
var con=document.getElementById(nam_ph).value;
document.getElementById("gal").insertAdjacentHTML("beforeend", con);
```

```
var b="dis"+(a+1);
if(document.getElementById(b))
  {
  document.getElementById(b).onclick=newph;
  }
return false;
}
</script>
</head>

<body>

<div id="gal">
<img src="city/0.jpg" alt="Фото">
<img src="city/1.jpg" alt="Фото">
<img src="city/2.jpg" alt="Фото"><br>
<a id="dis1" href="#">ЕЩЕ ФОТО...</a>
</div>

<textarea id="ph1" class="phot">
<img src="city/3.jpg" alt="Фото">
<img src="city/4.jpg" alt="Фото">
<img src="city/5.jpg" alt="Фото"><br>
<a id="dis2" href="#">ЕЩЕ ФОТО...</a>
</textarea>

<textarea id="ph2" class="phot">
<img src="city/6.jpg" alt="Фото">
<img src="city/7.jpg" alt="Фото">
<img src="city/8.jpg" alt="Фото">
</textarea>

</body>
</html>
```

При желании, читатели могут модернизировать функции **newph()** из обоих вариантов, рассмотренных в этом разделе. Вы, конечно, обратили внимание, что, добавляя новую порцию снимков, мы каждый раз заменяли текущие ссылки на новые. А если сделать ссылку постоянной и скрывать ее только после того, как на страницу загрузится последняя группа фотографий? Такой способ в некоторых случаях удобнее, так как упрощает разметку, исключая из нее лишние ссылки. Особенно это ощутимо, если скрытых блоков много – программисту меньше работы. В то же время свои достоинства есть и в рассмотренных мною примерах – они подойдут, если вы, допустим, хотите, чтобы каждая кнопка или ссылка имела персональное оформление (в моей практике такие

случаи были). Готовя примеры, я, после некоторого размышления, решил остановиться на более сложных вариантах, так как они дают программисту больше простора для творчества. Если же вы хотите попробовать модернизировать функции, думаю, вам не составит труда внести необходимые изменения.

Разбирая случаи отложенной загрузки, мы вели речь только о фотографиях. Понятно, что таким способом можно по частям загружать и другое содержимое: текст, формы, слои, код JavaScript и т. д. Исключение – поле textarea. Если его поместить внутрь другого textarea, это приведет к искажению разметки и возникновению ошибок. Возможных решений два: либо не делать этого, либо заменить вложенный тег textarea каким-то набором символов, которые будут снова преобразованы в тег textarea после того, как некая функция считает текст из основного поля.

И еще: хотя мы и подметили определенный недостаток в первом примере, это не отменяет его значимость. Если речь идет о пошаговой загрузке небольших объемов данных, вы вполне можете расположить их в слоях. Впрочем, мой совет – изучите какой-нибудь язык создания серверных программ – и тогда вы сможете писать скрипты, которые будут загружать страницы теми же методами, что используются в социальных сетях.

Вставка фото

До сих пор мы рассматривали случаи, когда список размещаемых фотографий был определен заранее. Обычно у каждой страницы есть основная тема, которую можно проиллюстрировать несколькими изображениями. Подбором снимков занимается либо сам разработчик (если он выполняет проект единолично), либо контент-менеджер (если сайт делает студия web-дизайна). После того, как страница скомпонована и опубликована в сети, ее содержание «фиксируется» до следующего обновления (которое может произойти очень нескоро). Таким образом, выбранные фото «прописываются» в HTML-коде опытными специалистами с применением соответствующих технологий и методов.

Но существует немало сайтов, где контент меняется ежеминутно, а наполнением страниц занимаются его многочисленные посетители. Речь, как вы уже догадались, идет о социальных сетях.

Типичное сообщение пользователя такой сети состоит из текста, а также одного или нескольких снимков. Добавить в сообщение фотографию можно, закачав ее со своего компьютера или смартфона. Этот случай мы пока не будем рассматривать – нам предстоит вспомнить о нем в третьей главе, когда речь пойдет об отправке форм. Есть еще другая возможность разместить картинку – выбрать ее из тех, что вы закачивали ранее для предыдущих постов или в альбомы. Такая функция реализована не во всех социальных сетях, но в российских – «Одноклассниках» и «ВКонтакте» – она присутствует.

В обеих упомянутых сетях форма создания очередной записи выглядит так: поле для ввода текста, а под ним пиктограмма фотоаппарата. Если ее нажать, откроется дополнительная вкладка, на которой вам будет предложено за-

грузить фотографию с компьютера или выбрать из уже имеющихся. Последние представлены на этой же вкладке, только в уменьшенном размере. Щелкаете на интересующем вас снимке – и он появляется в сообщении, а вкладка закрывается. Вот, собственно, и все.

Попробуем сделать нечто подобное и мы. Конечно, оформлен наш пример будет чуть проще, чем в социальных сетях, но суть останется та же.

Для вкладки можно выбрать разные «заготовки» (например, слой div), но мне захотелось воспользоваться элементом iframe, чтобы продемонстрировать читателям программное взаимодействие между двумя окнами. В главном окне у нас будет меню со списком альбомов и область вставки изображения, а во фрейме – уменьшенные копии снимков.

Первым делом разместим на странице список альбомов. У нас их два – «Города России» и «Города мира»:

```
<select id="va"><option selected value="city.html">Города России</option>
<option value="town.html">Города мира</option></select>
```

Рядом поставим кнопку "Смотреть", а под ней расположим контейнер для вставки изображения:

```
<input id="se" type="button" value="Смотреть"><br><br>
<span id="scru"></span>
```

Рис. 1_13. Страница выбора альбома

Добавим фрейм:

<iframe id="ram" src="city.html"></iframe>

Его настройки:

iframe {visibility: hidden; position: absolute; z-index: 55; left: 0px; top: 50px; width: 600px; height: 300px; box-shadow: 15px 15px 10px #000000; border: 8px solid #0000CC; border-radius: 8px;}

В исходном состоянии фрейм скрыт от пользователя.

Что у нас получилось, можно посмотреть на странице http://bookjs.ru/p6.html или на рисунке 1_13.

Теперь настало время сказать, что наш пример – особенный. Мы впервые создадим не одну, а две страницы. Для данной книги это уникальная ситуация, так как во всех остальных случаях мы размещали и в дальнейшем будем размещать код в одном документе.

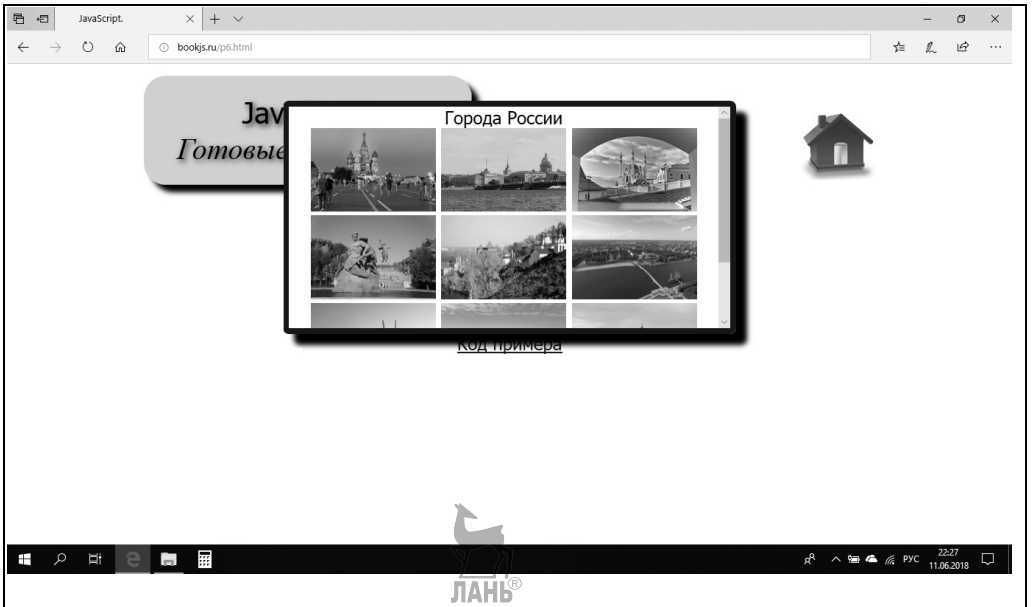

Рис. 1_14. Окно выбора фотографии

Вообще-то, в реальном примере я использовал три страницы: одну основную и две с фотографиями городов России и мира. Страницы с фото поочередно загружаются во фрейм. Так как они отличаются только адресами снимков, приводить код обеих документов не имеет смысла. Поэтому ограничимся описанием одного документа – city.html. Он содержит снимки городов, уже знакомые нам по предыдущим примерам этой главы:

**Города России
**
<img src="city/0.jpg" class="sma" alt="Фото">

```
<img src="city/1.jpg" class="sma" alt="Фото">
<img src="city/2.jpg" class="sma" alt="Фото"><br>
<img src="city/3.jpg" class="sma" alt="Фото">
<img src="city/4.jpg" class="sma" alt="Фото">
<img src="city/5.jpg" class="sma" alt="Фото"><br>
<img src="city/6.jpg" class="sma" alt="Фото">
<img src="city/7.jpg" class="sma" alt="Фото">
<img src="city/8.jpg" class="sma" alt="Фото">
```

Под снимками у нас есть ссылка

```
<a href="#">ЗАКРЫТЬ</a>
```

Правда, ссылка эта – чисто номинальная, только чтобы посетитель не терялся в догадках, каким образом закрыть фрейм. На самом деле любой клик не на фотографии, а на свободном поле фрейма приведет к тому, что **iframe** станет невидимым.

Добавим, что страница town.html имеет аналогичный код, только содержит адреса фотографий городов мира, расположенных в другой папке.

Проверим нашу программу в действии. Выберем в выпадающем списке альбом «Города России» и нажмем кнопку «Смотреть». После короткой паузы появится окно с фото. Как выглядит это окно, можно видеть на рисунке 1_14.

Рис. 1_15. Добавление фото на страницу

Предположим, мы решили прикрепить к нашей странице изображение Казани. Щелкаем на третьем снимке и видим, как фрейм исчез, а фото столицы Татарстана появилось на странице под списком альбомов. Передумали? Вновь жмем «Смотреть» и в открывшемся окне выбираем другой снимок. Например, столицы зимней Олимпиады 2014 года. Щелкаем на нем – фрейм вновь скроется, а вместо фотографии Казани на странице появится изображение Сочи (рис. 1_15). А если мы открыли окно, но, подумав, решили пока не делать выбор? Щелкните мышью в любом месте внутри рамки (но не на фото) и окно закроется.

Как работает эта «система»?

После загрузки основной страницы выполняется следующий блок инструкций:

```
window.onload=function()
{
var le=screen.width/2-308;
document.getElementById("ram").style.left=le+"px";
document.getElementById("se").onclick=see;
};
```

Как вы помните, **iframe** расположен у нас на 50 пикселей ниже верхней границы окна браузера. Теперь постараемся передвинуть его в центр по горизонтальной оси. Для этого сначала выясним ширину страницы. Затем, разделив ее значение на 2 и вычтя 308 пикселей (половина ширины **iframe** плюс толщина боковой рамки), мы найдем координату, поместив в которую верхний левый угол фрейма, мы добьемся желаемого результата:

```
var le=screen.width/2-308;
document.getElementById("ram").style.left=le+"px";
```

В последней строке блока

```
document.getElementById("se").onclick=see;
```

регистрируется обработчик для щелчка на кнопке «Смотреть». После ее нажатия будет запущена функция **see()**:

```
function see()
{
document.getElementById("ram").src=document.getElementById("va").value;
win-
dow.setTimeout('document.getElementById("ram").style.visibility="visible"',
500);
}
```

Свойству **src** фрейма присваивается адрес выбранного альбома с фотографиями и через 0,5 секунды **iframe** появляется на основной странице. Задержка в от-

крытии окна с альбомом необходима, чтобы фотографии по возможности успели загрузиться во фрейм до его появления (хотя при низкой скорости соединения этого времени может оказаться недостаточно).

Теперь перейдем на страницу city.html. Она содержит всего одну функцию, которая запускается при любом клике мышью на теле документа:

```
window.onclick=function(event)
{
if(event.target.tagName=="IMG")
  {
  parent.opt(event.target.src);
  }
else
  {
  parent.hid();
  }
};
```

Если щелчок выполнен на снимке, его адрес передается в функцию **opt()** (opt переводится как «выбирать»), которая «прописана» в основном документе. Если щелчок произошел на «свободном» пространстве или на ссылке «ЗАКРЫТЬ», вызывается функция **hid()** (от hidden) основного документа.

Объект parent *предоставляет доступ из дочернего фрейма в родительское окно. В некоторых источниках* ***parent*** *еще называют зарезервированным именем для обозначения ближайшего вмещающего окна.*

Вернемся на основную страницу. Посмотрим, что делает функция **opt()**:

```
function opt(snap)
{
document.getElementById("ram").style.visibility="hidden";
document.getElementById("scru").innerHTML='<img src="'+snap+'"
alt="Фото">';
}
```

Все максимально просто. Первая инструкция скрывает фрейм, а вторая помещает выбранное фото в контейнер **scru** (от английского scrutiny – внимательный осмотр), благодаря чему мы видим этот снимок на основной странице.

Функция **hid()** закрывает фрейм:

```
function hid()
{
document.getElementById("ram").style.visibility="hidden";
}
```

В полном листинге примера представлены два документа – основная страница и city.html:

---------- КОД ОСНОВНОЙ СТРАНИЦЫ ----------

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Вставка фото</title>
<style>
iframe {visibility: hidden; position: absolute; z-index: 55; left: 0px; top: 50px;
width: 600px; height: 300px; box-shadow: 15px 15px 10px #000000;
border: 8px solid #0000CC; border-radius: 8px;}
</style>

<script>
window.onload=function()
{
var le=screen.width/2-308;
document.getElementById("ram").style.left=le+"px";
document.getElementById("se").onclick=see;
};

function see()
{
document.getElementById("ram").src=document.getElementById("va").value;
window.setTimeout('document.getElementById("ram").style.visibility="visible"',
500);
}

function opt(snap)
{
document.getElementById("ram").style.visibility="hidden";
document.getElementById("scru").innerHTML='<img src="'+snap+'"
alt="Фото">';
}

function hid()
{
document.getElementById("ram").style.visibility="hidden";
}
</script>
</head>
```

```
<body>

<iframe id="ram" src="city.html"></iframe>

<select id="va"><option selected value="city.html">Города России</option>
<option value="town.html">Города мира</option></select>
<input id="se" type="button" value="Смотреть"><br><br>
<span id="scru"></span>

</body>
</html>
```

---------- КОД СТРАНИЦЫ city.html ----------

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Вставка фото</title>
<style>
body {text-align: center; margin: 0px}
.sma {width: 170px;}
</style>

<script>
window.onclick=function(event)
{
if(event.target.tagName=="IMG")
  {
  parent.opt(event.target.src);
  }
else
  {
  parent.hid();
  }
};
</script>
</head>

<body>

Города России<br>
<img src="city/0.jpg" class="sma" alt="Фото">
<img src="city/1.jpg" class="sma" alt="Фото">
<img src="city/2.jpg" class="sma" alt="Фото"><br>
```

```
<img src="city/3.jpg" class="sma" alt="Фото">
<img src="city/4.jpg" class="sma" alt="Фото">
<img src="city/5.jpg" class="sma" alt="Фото"><br>
<img src="city/6.jpg" class="sma" alt="Фото">
<img src="city/7.jpg" class="sma" alt="Фото">
<img src="city/8.jpg" class="sma" alt="Фото"><br>
<a href="#">ЗАКРЫТЬ</a>

</body>
</html>
```

Глава 2.
Меню – некоторые варианты

Перед вами самая короткая глава с довольно простыми примерами. Был период, когда авторы многих изданий, посвященных JavaScript, уделяли большое внимание рассмотрению весьма разнообразных вариантов интерактивных меню. Однако на сегодняшний день интерес к ним заметно снизился. «Навороты» уступили место практичности, особенно в нынешнее время повального увлечения лендингами. Конечно, традиционные меню не исчезли – они и сейчас являются неотъемлемой частью многих сайтов. Мы поговорим о двух самых распространенных случаях – выпадающих и анимированных меню.

Выпадающее меню

Сайты бывают разные. Одни состоят всего из несколько страниц, другие – из десятков или даже сотен. А в продвинутой социальной сети их миллионы.

О программах для социальных сетей речь в этой книге не идет – слишком большая и сложная тема. Простые сайты из 3–4 страниц тоже не попадают в сферу наших интересов – слишком элементарно. Поговорим о сайтах, где несколько десятков или несколько сотен страниц.

Важная особенность таких ресурсов – сложная многоуровневая структура. В большом проекте ссылок на внутренние страницы много, поэтому в один пункт меню может быть вложено другое меню (нижнего уровня).

Традиционно в большинстве сайтов (но не во всех) ссылки располагают либо вертикально, либо горизонтально.

Вертикальное меню:

Раздел 1
Раздел 2
Раздел 3
Раздел 4
Раздел 5

Если у какого-то раздела есть «подчиненные» страницы, то при нажатии на ссылку они тоже показываются в меню:

Раздел 1
Раздел 2
  Страница 1
  Страница 2
  Страница 3
Раздел 3

Раздел 4
Раздел 5

Вертикальное меню – достаточно простая «вещь», чаще всего не требующая применения JavaScript (за исключением случаев, когда разработчик хочет добавить интерактивности ссылкам).

Горизонтальное меню тоже не выглядит сложным:

Раздел 1 **Раздел 2** **Раздел 3** **Раздел 4** **Раздел 5**

Страница 1
Страница 2
Страница 3

Но это обманчивая простота. Когда в вертикальном меню появляются ссылки второго уровня, то блок меню растягивается, никак не влияя на основное содержимое, находящееся справа (чаще всего) или слева (реже). Если такой подход использовать при горизонтальном расположении ссылок, то появление пунктов второго уровня вызовет смещение контента вниз и появление пустот по сторонам от развернутой ссылки. В принципе избежать подобного эффекта можно, используя некоторые хитрости в разметке. Но! Во-первых, способ применим далеко не во всех случаях, а во-вторых, мы изучаем JavaScript, следовательно надо искать решение с использованием возможностей этого языка.

И такое решение есть. Более того, оно применяется очень часто очень многими разработчиками и уже очень давно. Называется это решение «выпадающее меню». Суть метода в следующем. Есть ссылка первого уровня. Принадлежащие ей дочерние ссылки объединяют в один блок и располагают его под родительской ссылкой «поверх» страницы. В результате контент остается на месте, а пустоты не образуются. Правда, открываясь, блок загораживает часть содержимого страницы, но это – на короткое время.

Мы рассмотрим два разных варианта выпадающих меню.

Начнем с более простого. Он предполагает, что пункты меню первого уровня жестко привязаны к конкретному месту на странице. Поэтому слой с меню второго уровня тоже имеет четкие координаты. Наша задача – составить программу, которая будет показывать выпадающее меню при наведении указателя мыши на ссылку и скрывать его, если посетитель воздержался от перехода по ссылке из основного и выпадающего меню.

Чтобы наш пример выглядел реалистично, мы создадим блок ссылок для раздела «ФОТО» демонстрационного сайта http://bookjs.ru/.

Главная ссылка:

```
<a id="lin" href="#">ФОТО</a>
```

Редкий случай, когда в стилевом оформлении наших примеров указан шрифт и его размер:

```
#lin {position: absolute; z-index: 51; left: 200px; top: 20px;
font-family: Tahoma; font-size: 24px;}
```

В данной ситуации назначение шрифта необходимо, чтобы ссылка не выглядела слишком примитивно.

Создадим обработчики для событий mouseover и mouseout:

```
window.onload=function()
{
…
document.getElementById("lin").onmouseover=vis;
document.getElementById("lin").onmouseout=cau;
…
};
```

Теперь создадим контейнер с меню второго уровня:

```
<div id="menu">
<a href="p1.html">просмотр фото</a> <br>
<a href="p2.html">выбор фото</a> <br>
<a href="p3.html">галерея</a> <br>
<a href="p4.html">слайд-шоу</a> <br>
<a href="p5.html">загрузка по частям</a></div>

#menu {position: absolute; z-index: 61; left: 190px; top: 50px;
visibility: hidden; background: #FFFFFF; border: 1px solid #0000CC;
padding: 15px; font-family: Tahoma; font-size: 24px;}
```

И обработчики для этого контейнера:

```
window.onload=function()
{
…
document.getElementById("menu").onmouseover=inp;
document.getElementById("menu").onmouseout=outp;
};
```

Посетитель может нажать ссылку «ФОТО». Но, поскольку такой страницы нет, в окне браузера ничего не изменится. Этот результат достигается за счет добавления обработчика клика, который возвращает значение **false**:

```
window.onload=function()
{
document.getElementById("lin").onclick=function()
```

```
  {
  return false;
  };
…
};
```

Страница, иллюстрирующая пример, располагается по адресу http://bookjs.ru/m1.html.

Блок ссылок в начальном состоянии не виден. Но при этом он находится на самом верху «пирамиды» из элементов страницы, так как имеет самый большой z-index: 61 (его значение выбрано произвольно). Когда слой видим, при наведении на него указателя мыши запускается функция **inp()** (имя образовано от input – вход). Когда указатель покидает слой, вызывается функция **outp()** (образовано от output – выход).

Перейдем к описанию и назначению функций. Но сначала создадим переменную **d**. Ориентируясь на ее значение, программа будет определять, где расположен указатель мыши – в области меню или вне:

```
var d=0;
```

Итак, при наведении указателя мыши на надпись «ФОТО» запускается функция **vis()** (от visible). В ней всего одна инструкция:

```
function vis()
{
document.getElementById("menu").style.visibility="visible";
}
```

Слой с меню становится видимым (рис. 2_1). Клиент посмотрит на список ссылок второго уровня и либо решит перейти на одну из страниц, либо не станет этого делать. В обоих случаях указатель мыши переместится с текста «ФОТО». В результате запустится функция **cau()** (от caught – задержанный):

```
function cau()
{
window.setTimeout(sea, 200);
}
```

Ее назначение – выдержать паузу 0,2 секунды и запустить функцию **sea()** (от английского seat – размещаться):

```
function sea()
{
if(d==0)
```

```
  {
  document.getElementById("menu").style.visibility="hidden";
  }
}
```

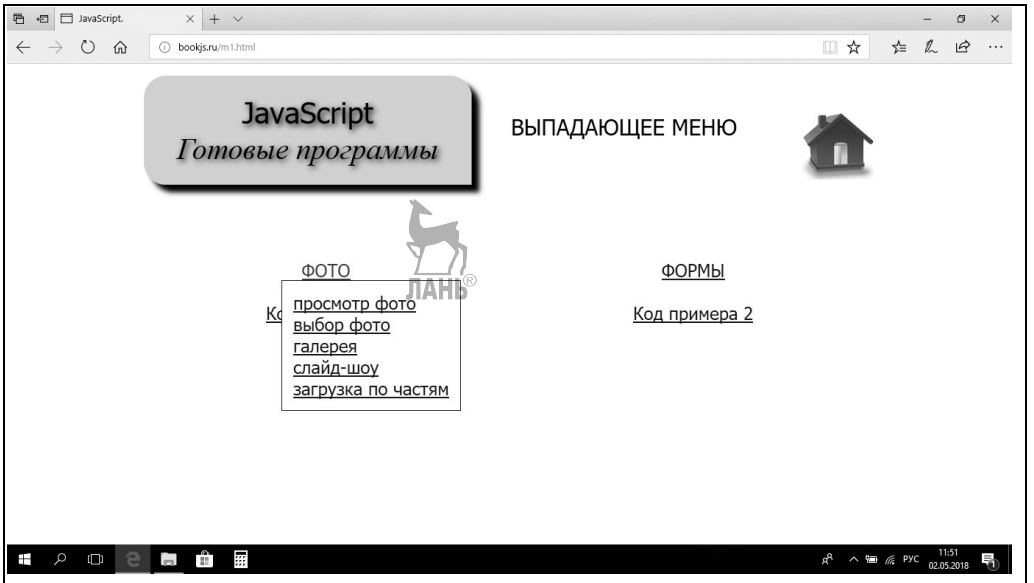

Рис. 2_1. Пример выпадающего меню

Если посетитель «ушел» из меню совсем, то значение переменной **d** не изменится и блок ссылок второго уровня скроется:

```
document.getElementById("menu").style.visibility="hidden";
```

Если посетитель сместил указатель мыши на выпадающее меню, то сработают две функции:

```
function inp()
{
d=1;
}
```

и

```
function cau()
{
window.setTimeout(sea, 200);
}
```

Переменная **d** получит значение 1, а так как функция **sea()** будет запущена с задержкой, то ее выполнение начнется уже после того, как изменилась **d**. В результате условие **d==0** окажется ложными и выпадающее меню останется на месте. Оно скроется только после того, как посетитель «покинет» блок ссылок:

```
function outp()
{
d=0;
window.setTimeout(sea, 200);
}
```

В этой части программы переменной **d** будет возвращено начальное значение, условие **d==0** в функции **sea()** вновь будет истинным и опять выполнится инструкция

```
document.getElementById("menu").style.visibility="hidden";
```

Не самая сложная программа среди тех, что мы рассматриваем в книге:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Выпадающее меню 1</title>
<style>
#lin {position: absolute; z-index: 51; left: 200px; top: 20px;
font-family: Tahoma; font-size: 24px;}
#menu {position: absolute; z-index: 61; left: 190px; top: 50px;
visibility: hidden; background: #FFFFFF; border: 1px solid #0000CC;
padding: 15px; font-family: Tahoma; font-size: 24px;}
</style>

<script>
window.onload=function()
{
document.getElementById("lin").onclick=function()
  {
  return false;
  };

document.getElementById("lin").onmouseover=vis;
document.getElementById("lin").onmouseout=cau;
document.getElementById("menu").onmouseover=inp;
document.getElementById("menu").onmouseout=outp;
};
```

```
var d=0;

function vis()
{
document.getElementById("menu").style.visibility="visible";
}

function inp()
{
d=1;
}

function outp()
{
d=0;
window.setTimeout(sea, 200);
}

function cau()
{
window.setTimeout(sea, 200);
}

function sea()
{
if(d==0)
  {
  document.getElementById("menu").style.visibility="hidden";
  }
}
</script>
</head>

<body>

<a id="lin" href="#">ФОТО</a>

<div id="menu">
<a href="p1.html">просмотр фото</a> <br>
<a href="p2.html">выбор фото</a> <br>
<a href="p3.html">галерея</a> <br>
<a href="p4.html">слайд-шоу</a> <br>
<a href="p5.html">загрузка по частям</a></div>

</body>
```

```
</html>
```

Этот вариант хорош для случаев, когда ваш проект стабилен и количество пунктов в меню первого уровня со временем не меняется. Иная ситуация, когда сайт находится в активном развитии и сегодня в горизонтальном меню, скажем, 5 ссылок, а завтра – 6, 7 или 8. Очевидно, что добавляя новые пункты, придется каждый раз менять расположение блоков меню второго уровня, чтобы место их появления совпадало с расположением родительских ссылок. Думаю, читатели согласятся, что это слишком хлопотное занятие. В таком случае лучшее решение – использовать выпадающее меню, ориентированное на место клика по ссылке.

Такой пример в нашем арсенале тоже есть. Посмотреть его можно на той же странице сайта.

В новом варианте мы для разнообразия изменили адрес основной ссылки и адреса ссылок меню второго уровня:

```
<a id="lin" href="#">ФОРМЫ</a>

<div id="menu">
<a href="f1.html">появление формы</a> <br>
<a href="f2.html">Ajax</a> <br>
<a href="f3.html">таймер загрузки</a> <br>
<a href="f4.html">подсказки</a> <br>
<a href="f5.html">проверка формы</a></div>
```

Впрочем, адреса ссылок принципиальной роли не играют. Важнее другое изменение – исчезла инструкция, которая делала видимым выпадающее меню при наведении указателя мыши на родительскую ссылку. Вместо этого появился вызов новой функции **orient()**, который происходит при клике на главной ссылке:

```
window.onload=function()
{
document.getElementById("lin").onclick=orient;
document.getElementById("lin").onmouseout=cau;
document.getElementById("menu").onmouseover=inp;
document.getElementById("menu").onmouseout=outp;
};
```

Причем мы не просто вызываем обработчик, а передаем ему в качестве аргумента объект события, благодаря чему можем определить место щелчка:

```
function orient(event)
{
…
}
```

Соответственно нашим новым задачам немного изменились настройки стилей:

```
#lin {position: absolute; z-index: 51; left: 200px; top: 20px;
font-family: Tahoma; font-size: 24px;}
#menu {position: absolute; z-index: 61; visibility: hidden; background: #FFFFFF;
border: 1px solid #0000CC; padding: 15px; font-family: Tahoma; font-size: 24px;}
```

Поскольку исходное позиционирование в новом варианте не играет роли, мы избавились от него в описании настроек блока **menu**. Теперь, как мы уже выяснили, определением места, где должно появиться выпадающее меню, «занимается» функция **orient()**:


```
function orient(event)
{
var e=event;
var x=e.clientX-50;
var y=e.clientY;
document.getElementById("menu").style.left=x+"px";
document.getElementById("menu").style.top=y+"px";
document.getElementById("menu").style.visibility="visible";
}
```

Создаем переменную **e** и присваиваем ей объект события:

```
var e=event;
```

Определяем координаты щелчка:

```
var x=e.clientX-50;
var y=e.clientY;
```

К координате **x** добавлено небольшое смещение влево, иначе левая граница меню окажется строго под местом щелчка, что будет выглядеть не очень естественно.

Свойства clientX и clientY *хранят информацию о вертикальном и горизонтальном смещении указателя мыши относительно левого верхнего угла окна браузера (в пикселях). Проще говоря, они предоставляют координаты мыши.*

Устанавливаем положение блока ссылок:

```
document.getElementById("menu").style.left=x+"px";
document.getElementById("menu").style.top=y+"px";
```

И, наконец, открываем меню:

```
document.getElementById("menu").style.visibility="visible";
```

Результат показан на рисунке 2_2.

Остальная часть программы – функции **inp()**, **outp()** и **cau()** – не изменились.

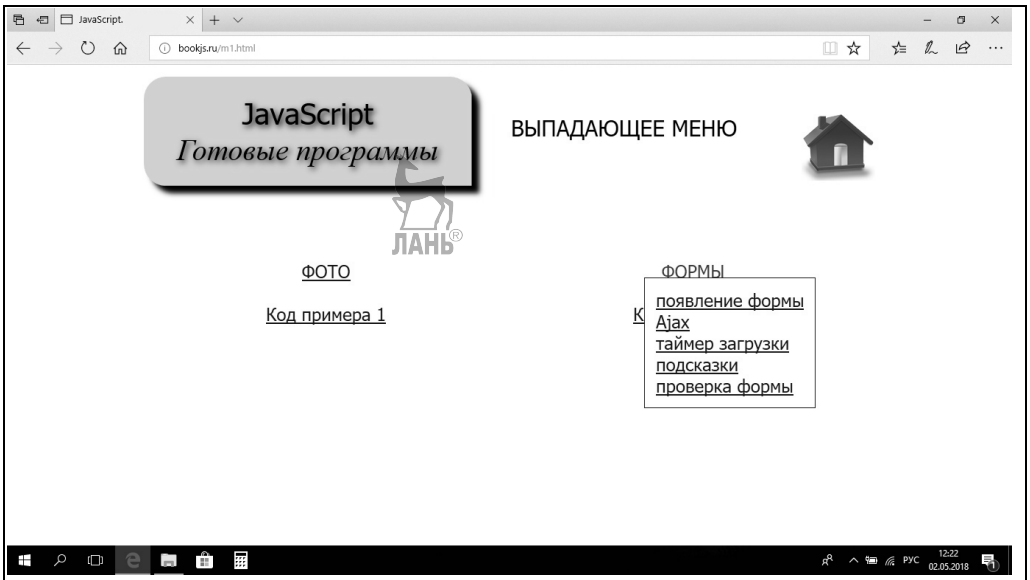

Рис. 2_2. По щелчку на правой ссылке появляется меню раздела «ФОРМЫ»

Листинг второго варианта:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Выпадающее меню 2</title>
<style>
#lin {position: absolute; z-index: 51; left: 200px; top: 20px;
font-family: Tahoma; font-size: 24px;}
#menu {position: absolute; z-index: 61; visibility: hidden; background:
#FFFFFF;
border: 1px solid #0000CC; padding: 15px; font-family: Tahoma; font-size:
24px;}
```

```
</style>

<script>
window.onload=function()
{
document.getElementById("lin").onclick=orient;
document.getElementById("lin").onmouseout=cau;
document.getElementById("menu").onmouseover=inp;
document.getElementById("menu").onmouseout=outp;
};

function orient(event)
{
var e=event;
var x=e.clientX-50;
var y=e.clientY;
document.getElementById("menu").style.left=x+"px";
document.getElementById("menu").style.top=y+"px";
document.getElementById("menu").style.visibility="visible";
return false;
}

var d=0;

function inp()
{
d=1;
}

function outp()
{
d=0;
window.setTimeout(sea, 200);
}

function cau()
{
window.setTimeout(sea, 200);
}

function sea()
{
if(d==0)
  {
  document.getElementById("menu").style.visibility="hidden";
```

```
  }
}
</script>
</head>

<body>

<a id="lin" href="#">ФОРМЫ</a>

<div id="menu">
<a href="f1.html">появление формы</a> <br>
<a href="f2.html">Ajax</a> <br>
<a href="f3.html">таймер загрузки</a> <br>
<a href="f4.html">подсказки</a> <br>
<a href="f5.html">проверка формы</a></div>

</body>
</html>
```

Мы разбирали ситуацию с выпадающим меню только для одной ссылки. Если ссылок с вложенными меню несколько, вам будет нужно усовершенствовать программу (точнее, ту из двух рассмотренных, что вы решите использовать в своем проекте). При определении, какая ссылка должна раскрываться, можно действовать теми же методами, что мы использовали для «вычисления» загружаемой фотографии в первой главе (разделы «Выбор фото» или «Галерея»). В этом вам поможет объект события **event**: сначала надо узнать, на какой ссылке оказался указатель мыши или выполнен клик, а затем передать соответствующий адрес, например, в виде аргумента, в функции. После экспериментов с фотографиями добавление аналогичных возможностей в программу выпадающего меню не должно вызвать у вас затруднений.

Анимированное меню

С анимированными меню сталкиваются все пользователи Интернета. Наверняка вы не раз видели, как при наведении указателя мыши на ссылку она меняет цвет или у нее появляется нижнее подчеркивание (иногда, наоборот, подчеркивание исчезает). Это – наиболее простые элементы анимации. Обычно они получаются без использования JavaScript. Необходимо лишь указать желаемые параметры в таблице стилей. Например, так:

```
<style>
a {color: #0000CC;}
a:hover {color: #CC0000;}
</style>
```

Была ссылка синего цвета, навели указатель мыши – стала ссылка красной.

Но во многих случаях анимацию делают более сложной. Тут на помощь как раз и приходит JavaScript. Достигаемые эффекты иногда оказываются весьма внушительными и производят довольно сильное впечатление. Мы не станем гнаться за сногсшибательными результатами, а рассмотрим достаточно простые приемы анимирования меню. Получив начальный опыт, заинтересованные читатели смогут продолжить эксперименты. Кстати, некоторые полезные подсказки на эту тему мы дадим еще и в последней главе.

Как и в предыдущем разделе, приведем два примера. На этот раз у нас будет меню с использованием графических элементов и без таковых. Оба варианта представлены на сайте поддержки на странице http://bookjs.ru/m2.html. Также вы можете видеть их на рисунке 2_3.

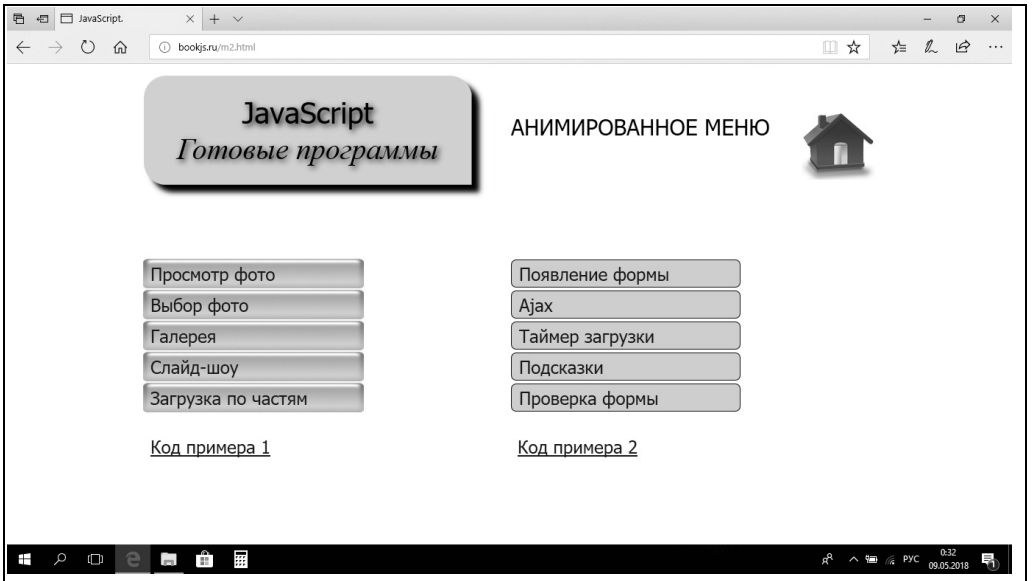

Рис. 2_3. Примеры анимированного меню

Начнем с графического меню. Оно разместилось в левом столбце. Каждый пункт меню состоит из двух элементов: рисованной кнопки и текстовой ссылки. Рабочим элементом является ссылка. Кнопка используется только в качестве декорации. В исходном состоянии ссылки синие, а кнопки – бледно-голубого цвета. Теперь наведите указатель мыши на одну из ссылок – она станет красной, а кнопка – оранжевой. Если отвести мышь в сторону, ссылка и кнопка вернутся в первоначальное состояние.

Изменение цвета ссылки достигается способом, продемонстрированным выше. Мы только добавили в стили свойство **text-decoration: none**, чтобы текст не был подчеркнут:

```
a {color: #0000CC; text-decoration: none;}
a:hover {color: #CC0000;}
```

Кнопками управляет небольшая программа, написанная на JavaScript.

Но прежде чем мы начнем разбираться в принципе ее работы, разместим на странице необходимые элементы навигации:

```
<table id="tab">
<tr><td id="tbo1"><a href="p1.html" id="lin1">просмотр
фото</a></td></tr>
<tr><td id="tbo2"><a href="p2.html" id="lin2">выбор фото</a></td></tr>
<tr><td id="tbo3"><a href="p3.html" id="lin3">галерея</a></td></tr>
<tr><td id="tbo4"><a href="p4.html" id="lin4">слайд-шоу</a></td></tr>
<tr><td id="tbo5"><a href="p5.html" id="lin5">загрузка по
частям</a></td></tr>
</table>
```

Уже не в первый раз мы используем таблицу для упрощения компоновки элементов. В пяти ячейках таблицы расположились ссылки. Изображения кнопок – это фоновые рисунки ячеек:

```
table {width: 304px; font-family: Tahoma; font-size: 24px;}
td {height: 38px; text-align: left; background-image: url(img/but1.jpg);
background-repeat: no-repeat; padding-left: 10px;}
```

Значение **padding-left: 10px** устанавливает отступ ссылки от левой границы блока меню.

Задача нашей программы – поменять фоновое изображение в ячейке, когда указатель мыши наводится на соответствующую ссылку, и вернуть исходный фон, когда указатель покинул ссылку.

Как всегда, регистрируем обработчики событий mouseover и mouseout:

```
window.onload=function()
{
document.getElementById("tab").onmouseover=unc;
document.getElementById("tab").onmouseout=clo;
};
```

Создаем переменную, которая будет хранить ID задействованной ссылки:

```
var h;
```

Пишем функцию обработки события mouseover:

```
function unc(event)
```

```
{
if(event.target.id.indexOf("lin")==0)
  {
  var p=event.target.id.split("lin");
  h="tbo"+p[1];
  document.getElementById(h).style.backgroundImage="url(img/but2.jpg)";
  }
}
```

Первый оператор сравнения

if(event.target.id.indexOf("lin")==0)

выясняет, произошло ли событие на ссылке. Если да, то следующий шаг – определить, на какой:

event.target.id

Затем надо вычислить цифровой индекс **id** ссылки:

var p=event.target.id.split("lin");

Разбивая **id** по шаблону **lin**, мы получаем массив **p**, состоящий из двух элементов: p[0] и **p[1]**. Первый из них – p[0] – пустой, а второй – **p[1]** – содержит цифровую часть **id** ссылки. Нам осталось узнать, в какой ячейке произошло событие mouseover:

h="tbo"+p[1];

Теперь в этой ячейке можно заменить голубое фоновое изображение на оранжевое (рис. 2_4):

document.getElementById(h).style.backgroundImage="url(img/but2.jpg)";

Если указатель мыши покинул ссылку, запускается функция **clo(),** которая получила свое имя от сокращения термина close:

```
function clo(event)
{
if(event.target.id.indexOf("lin")==0)
  {
  document.getElementById(h).style.backgroundImage="url(img/but1.jpg)";
  }
}
```

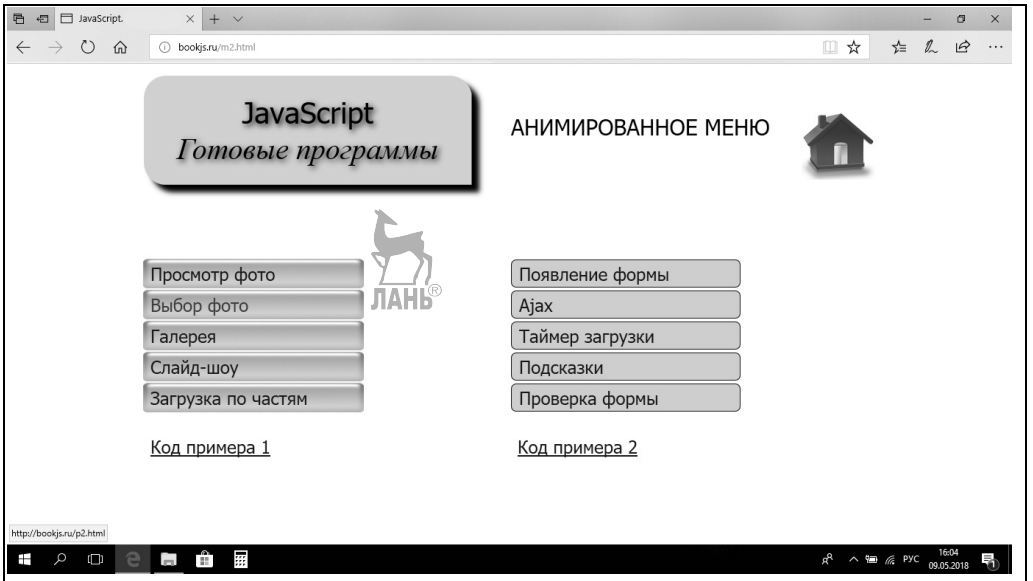

Рис. 2_4. Смена фонового изображения в пункте меню

Опять убеждаемся, что событие произошло на ссылке

if(event.target.id.indexOf("lin")==0)

после чего возвращаем первоначальный фоновый рисунок соответствующей ячейке таблицы:

document.getElementById(h).style.backgroundImage="url(img/but1.jpg)";

Так как ID ссылки и ячейки мы уже определили в функции **unc()** (от unclose – открывать), повторно вычислять их нет необходимости.

Листинг программы выглядит несложным:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Анимированное меню 1</title>
<style>
a {color: #0000CC; text-decoration: none;}
a:hover {color: #CC0000;}
table {width: 304px; font-family: Tahoma; font-size: 24px;}
td {height: 38px; text-align: left; background-image: url(img/but1.jpg); back-
ground-repeat: no-repeat; padding-left: 10px;}
</style>
```

```
<script>
window.onload=function()
{
document.getElementById("tab").onmouseover=unc;
document.getElementById("tab").onmouseout=clo;
};

var h;

function unc(event)
{
if(event.target.id.indexOf("lin")==0)
  {
  var p=event.target.id.split("lin");
  h="tbo"+p[1];
  document.getElementById(h).style.backgroundImage="url(img/but2.jpg)";
  }
}

function clo(event)
{
if(event.target.id.indexOf("lin")==0)
  {
  document.getElementById(h).style.backgroundImage="url(img/but1.jpg)";
  }
}
</script>
</head>

<body>

<table id="tab">
<tr><td id="tbo1"><a href="p1.html" id="lin1">просмотр
фото</a></td></tr>
<tr><td id="tbo2"><a href="p2.html" id="lin2">выбор фото</a></td></tr>
<tr><td id="tbo3"><a href="p3.html" id="lin3">галерея</a></td></tr>
<tr><td id="tbo4"><a href="p4.html" id="lin4">слайд-шоу</a></td></tr>
<tr><td id="tbo5"><a href="p5.html" id="lin5">загрузка по
частям</a></td></tr>
</table>

</body>
</html>
```

Приступим к рассмотрению второго варианта анимированного меню. Оно разместилось в правом столбце. Здесь каждый пункт состоит из трех элементов:

фоновой подложки, контура кнопки и текстовой ссылки. Все остальное устроено, как и в первом варианте. Рабочий элемент – ссылка. Кнопка – только часть декоративного оформления. В исходном состоянии ссылки синие, а кнопки – бледно-голубого цвета. При наведении указателя мыши на ссылку она становится красной, а кнопка – оранжевой.

Для разнообразия поменяем меню:

```
<table id="tab">
<tr><td id="tbo1"><a href="f1.html" id="lin1">появление
формы</a></td></tr>
<tr><td id="tbo2"><a href="f2.html" id="lin2">Ajax</a></td></tr>
<tr><td id="tbo3"><a href="f3.html" id="lin3">таймер
загрузки</a></td></tr>
<tr><td id="tbo4"><a href="f4.html" id="lin4">подсказки</a></td></tr>
<tr><td id="tbo5"><a href="f5.html" id="lin5">проверка
формы</a></td></tr>
</table>
```

Новая таблица стилей будет такой:

```
<style>
a {color: #0000CC; text-decoration: none;}
a:hover {color: #CC0000;}
table {width: 304px; font-family: Tahoma; font-size: 24px;}
td {height: 38px; text-align: left; background: #B4D3FF;
width: 302px; height: 36px; border: 1px solid #0000CC;
border-radius: 7px; line-height: 36px; padding-left: 10px;}
</style>
```

Версия функции **unc()** для второго варианта:

```
function unc(event)
{
if(event.target.id.indexOf("lin")==0)
  {
  var p=event.target.id.split("lin");
  h="tbo"+p[1];
  document.getElementById(h).style.background="#FFC1A0";
  document.getElementById(h).style.border="1px solid #CC0000";
  }
}
```

Теперь в ней две новые инструкции:

```
document.getElementById(h).style.background="#FFC1A0";
document.getElementById(h).style.border="1px solid #CC0000";
```

В первой строке голубой фон меняется на оранжевый. Во второй – рамка синего цвета становится красной (рис. 2_5).

В функции **clo()** появились две инструкции, возвращающие кнопкам исходный внешний вид:

```
document.getElementById(h).style.background="#B4D3FF";
document.getElementById(h).style.border="1px solid #0000CC";
```

Листинг второго примера:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Анимированное меню 2</title>
<style>
a {color: #0000CC; text-decoration: none;}
a:hover {color: #CC0000;}
table {width: 304px; font-family: Tahoma; font-size: 24px;}
td {height: 38px; text-align: left; background: #B4D3FF;
width: 302px; height: 36px; border: 1px solid #0000CC;
border-radius: 7px; line-height: 36px; padding-left: 10px;}
</style>

<script>
window.onload=function()
{
document.getElementById("tab").onmouseover=unc;
document.getElementById("tab").onmouseout=clo;
};

var h;

function unc(event)
{
if(event.target.id.indexOf("lin")==0)
  {
  var p=event.target.id.split("lin");
  h="tbo"+p[1];
  document.getElementById(h).style.background="#FFC1A0";
  document.getElementById(h).style.border="1px solid #CC0000";
  }
}

function clo(event)
```

```
{
if(event.target.id.indexOf("lin")==0)
  {
  document.getElementById(h).style.background="#B4D3FF";
  document.getElementById(h).style.border="1px solid #0000CC";
  }
}
</script>
</head>

<body>

<table id="tab">
<tr><td id="tbo1"><a href="f1.html" id="lin1">появление
формы</a></td></tr>
<tr><td id="tbo2"><a href="f2.html" id="lin2">Ajax</a></td></tr>
<tr><td id="tbo3"><a href="f3.html" id="lin3">таймер
загрузки</a></td></tr>
<tr><td id="tbo4"><a href="f4.html" id="lin4">подсказки</a></td></tr>
<tr><td id="tbo5"><a href="f5.html" id="lin5">проверка
формы</a></td></tr>
</table>

</body>
</html>
```

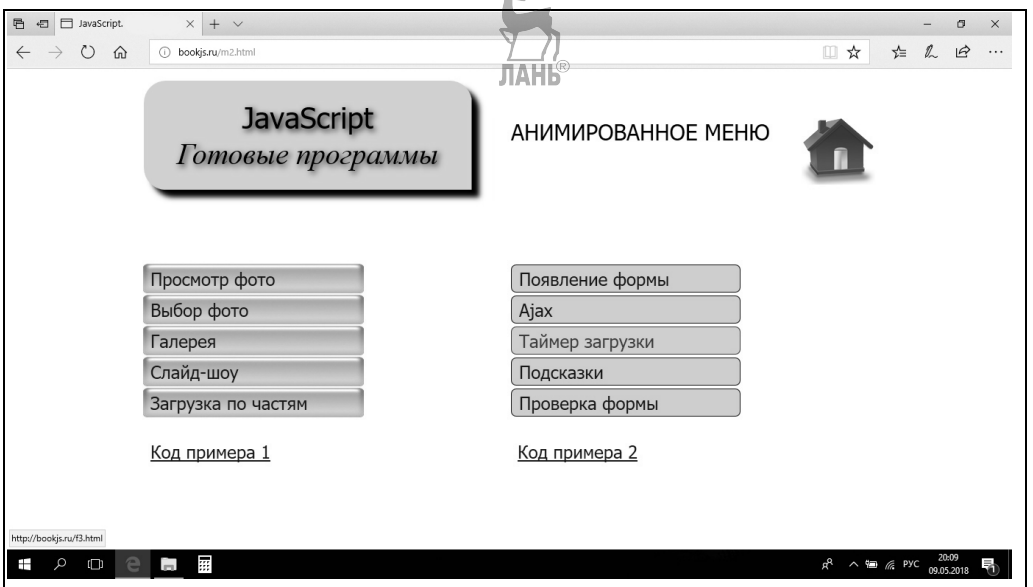

Рис. 2_5. Второй вариант анимированного меню

Подводя итог, можно сказать, что оба примера работают похожим образом: в первом случае мы меняем фоновый рисунок, во втором – цвет фона и рамки кнопки.

Второй пример в нашем исполнении выглядит довольно просто. Чтобы сделать его более «дизайнерским», можно указать градиент для свойства background:

background: linear-gradient(*значения***)**

Если правильно подобрать 6–8 цветов и указать их параметры в настройках градиента, можно получить кнопки, мало в чем уступающие рисованным.

Глава 3
Формы и работа с ними

Едва ли не все сайты имеют формы пересылки сообщений. Одни необходимы для связи с представителями компаний, другие нужны для оформления заявок на какие-либо услуги, третьи используются для отправки заказов из Интернет-магазинов. Обычно посетитель заполняет форму, нажимает кнопку и данные передаются на сервер специальной программе, которая обрабатывает информацию. Затем она записывается в файл или пересылается почтой администратору. После этого серверная программа передает в браузер клиента сообщение об успешном завершении процесса (или об ошибке, если посетитель ввел некорректные данные). В этой главе мы рассмотрим все этапы работы с формами непосредственно в браузере, не касаясь серверной части.

Появление формы

Обычно под формы отводится отдельная страница. Чаще всего они располагаются в разделе «Контакты» или «Оформление заказа». Бывает, что форма находится прямо в тексте страницы, если это обусловлено логикой ее применения (например, форма проверки штрафов на сайте ГИБДД). В последнем случае используют два подхода: посетитель сразу видит поля, необходимые для заполнения, либо они открываются при нажатии на специальную кнопку или ссылку. Мы рассмотрим второй вариант.

Представим, что у нас есть форма, состоящая из четырех полей – «Ваше имя», «Телефон», «E-mail», «Сообщение» – и кнопка отправки данных. Посетитель может передать нам какую-то информацию, а может не делать этого. Чтобы не перегружать страницу, скроем форму, оставив только кнопку или ссылку «Отправить сообщение». Первое нажатие на нее приведет к появлению формы, повторное – к скрытию. По-моему, вполне удобно.

Рассмотренный нами пример можно увидеть в действии на странице http://bookjs.ru/f1.html или на рисунках 3_1 и 3_2.

Как всегда, создаем простую HTML-страницу и размещаем в ней форму и кнопку «Отправить сообщение»:

```
<input type="button" id="bu" value="Отправить сообщение">

<div id="fo"><br><br>
<form>Ваше имя: <input type="text" name="F1"><br>
Телефон: <input type="text" name="F2"><br>
E-mail: <input type="text" name="F3"><br>
Сообщение: <textarea name="F4"></textarea><br>
<input type="button" value="Отправить"></form></div>
```

Свойство **display** слоя с формой:

#fo {display: none;}

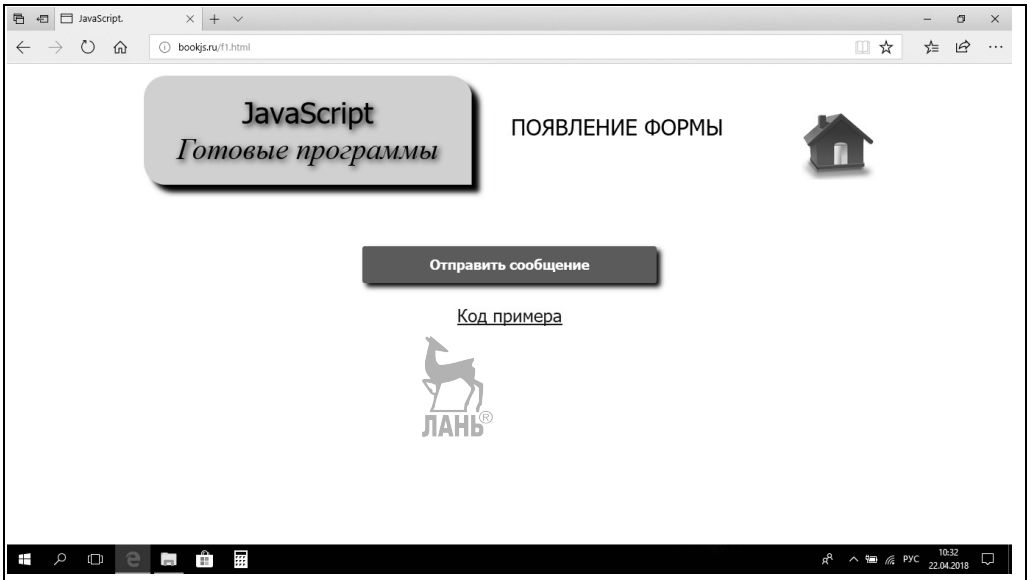

Рис. 3_1. Исходное состояние страницы: форма скрыта от посетителя

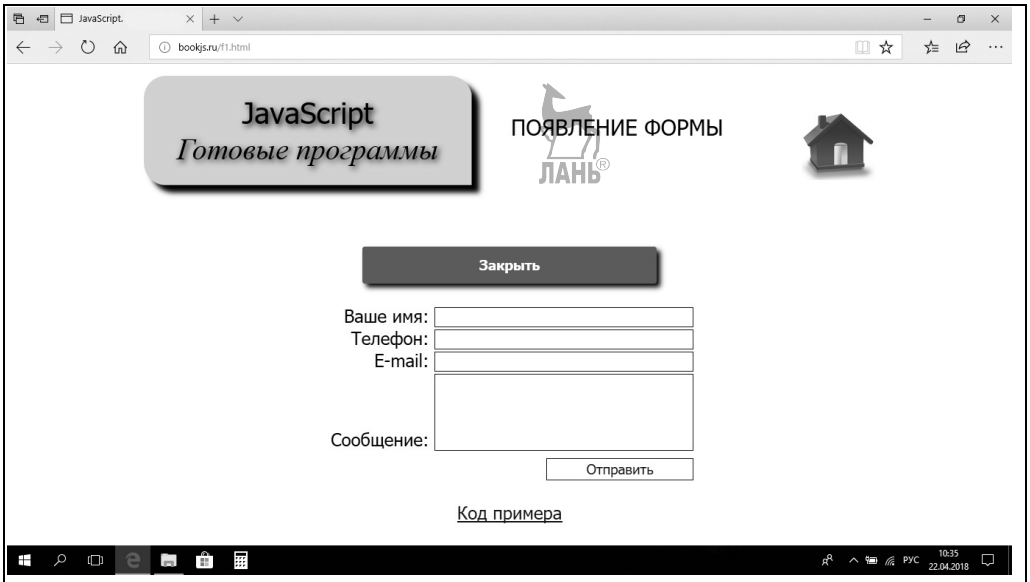

Рис. 3_2. Посетитель нажал кнопку «Отправить сообщение» и форма появилась на странице

В исходном состоянии контейнер **div** с формой скрыт, так как его свойству **display** присвоено значение **none**. Посетитель видит только кнопку. Мы добавили ей персональный ID, чтобы можно было на программном уровне обращаться к ее свойствам и менять надпись.

Размещая форму, мы не указали параметры method и action. Сейчас они не нужны – отправлять сообщения мы пока не планируем. У нас иная задача.

Функция **ope()** (от английского open), управляющая появлением и скрытием формы, вызывается нажатием на кнопку:

```
window.onload=function()
{
document.getElementById("bu").onclick=ope;
};
```

В процессе загрузки страницы мы создали переменную **i**. Это индикатор состояния формы. Начальное значение переменной – 0.

Ниже приведем код функции:

```
function ope()
{
if(i==0)
  {
  document.getElementById("bu").value="Закрыть";
  document.getElementById("fo").style="display: inline";
  i=1;
  }
else
  {
  document.getElementById("bu").value="Отправить сообщение";
  document.getElementById("fo").style="display: none";
  i=0;
  }
}
```

Первый клик на кнопке приводит к выполнению первого блока инструкций:

```
document.getElementById("bu").value="Закрыть";
document.getElementById("fo").style="display: inline";
i=1;
```

У кнопки меняется надпись, форма появляется на странице, а переменной **i** присваивается значение 1. Если вновь нажать кнопку, то выполнится второй блок инструкций:

```
document.getElementById("bu").value="Отправить сообщение";
document.getElementById("fo").style="display: none";
i=0;
```

Кнопке будет возвращена надпись «Отправить сообщение», форма скроется, а переменная **i** примет значение 0. Как видите, индикатор состояния формы меняет свое значение при каждом клике, поэтому открывать и закрывать форму можно бесконечно. Каждый нечетный клик ведет к появлению формы, каждый четный – к ее скрытию.

Соберем отдельные части в единое целое:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Появление формы</title>
<style>
#fo {display: none;}
</style>

<script>
window.onload=function()
{
document.getElementById("bu").onclick=ope;
};

var i=0;

function ope()
{
if(i==0)
  {
  document.getElementById("bu").value="Закрыть";
  document.getElementById("fo").style="display: inline";
  i=1;
  }
else
  {
  document.getElementById("bu").value="Отправить сообщение";
  document.getElementById("fo").style="display: none";
  i=0;
  }
}
</script>
</head>
```

```
<body>

<input type="button" id="bu" value="Отправить сообщение">

<div id="fo"><br><br>
<form>Ваше имя: <input type="text" name="F1"><br>
Телефон: <input type="text" name="F2"><br>
E-mail: <input type="text" name="F3"><br>
Сообщение: <textarea name="F4"></textarea><br>
<input type="button" value="Отправить"></form></div>

</body>
</html>
```

Кстати, внимательный читатель наверняка обратил внимание, что для управления формой мы применили метод, очень схожий с тем, что использовали при загрузке фотографий по частям.

Первый пример в данной главе был весьма простым. Посмотрим, что будет дальше.

Ajax

Напомню, что Ajax расшифровывается как «Асинхронный JavaScript и XML» и представляет собой передовую технологию взаимодействия между браузером и сервером.

Как происходила отправка данных из форм раньше? Клиент отсылал информацию, а сервер передавал в браузер новую HTML-страницу с сообщением типа «Ваша заявка зарегистрирована» или «Вы указали некорректные сведения». Ключевой момент ранних технологий – загрузка новой страницы вместо исходной. Это увеличивает время ожидания ответа сервера, повышает Интернет-трафик. При использовании технологии Ajax перезагрузка страницы не происходит – к ней просто добавляются поступившие от сервера данные. Разница между отправкой сообщений с перезагрузкой страницы и без нее весьма ощутима, особенно, когда на страницу надо добавить всего-навсего короткий текст «Ваша заявка зарегистрирована».

Учитывая, что современные посетители сайтов любят, чтобы все делалось как можно быстрее, автор советует web-программистам использовать Ajax всегда, когда это возможно.

Попутно заметим, что применение этой технологии не ограничивается только отправкой форм. Например, добавление контента на страницу при ее прокрутке, как это происходит в социальных сетях и на многих сайтах, тоже реализовано с помощью Ajax.

Было бы непростительно, если бы мы прошли мимо такой передовой методики и не научились отправлять формы с помощью Ajax. Поэтому следующий пример будет посвящен этой технологии.

Сначала примем решение, что для наших экспериментов вполне подойдет форма из предыдущего раздела. Только для упрощения кода мы откажемся от механизма ее появления и скрытия. Будем считать, что поля формы постоянно находятся на странице:

```
<form>Ваше имя: <input type="text" name="F1" id="F1"><br>
Телефон: <input type="text" name="F2" id="F2"><br>
E-mail: <input type="text" name="F3" id="F3"><br>
Сообщение: <textarea name="F4" id="F4"></textarea><br>
<input id="bu" type="button" value="Отправить"></form>
```

В форму мы внесли два исправления – всем полям присвоили ID, а кнопке **button** добавили вызов функции отправки данных **ret()** (от английского retrive, что в переводе означает «возвращать»):

```
window.onload=function()
{
document.getElementById("bu").onclick=ret;
};
```

Ниже под кнопкой «Отправить» расположим два контейнера, в которых будет отображаться процесс отправки данных и результат ответа сервера:

```
<span id="att1"><br>СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...</span>
<span id="att2"><br>СООБЩЕНИЕ ДОСТАВЛЕНО</span>
```

Свойства контейнеров:

```
#att1 {display: none; color: #CC0000;}
#att2 {display: none; color: #0000CC;}
```

В исходном состоянии обе надписи скрыты.

Перейдем к описанию функции **ret()**:

```
function ret()
{
var re=new XMLHttpRequest();
re.onload=show;
var one=document.getElementById("F1").value;
var two=document.getElementById("F2").value;
var tri=document.getElementById("F3").value;
var fou=document.getElementById("F4").value;
```

```
var res="record.php?F1="+one+"&F2="+two+"&F3="+tri+"&F4="+fou;
re.open("GET", res, true);
re.send();
document.getElementById("att1").style.display="inline";
}
```

В первой строке функции создается новый объект **XMLHttpRequest**, с помощью которого происходит взаимодействие web-страницы и сервера. Вторая строка сообщает программе, что при возвращении ответа от сервера должна запускаться функция **show()**. В следующих четырех строках значение полей формы присваиваются переменным. Затем формируется адрес, по которому произойдет обращение к серверной программе. Мы планируем передавать информацию методом GET, поэтому вслед за именем вызываемого скрипта **record.php** идет перечисление имен полей формы и их значений. Получившаяся строка присваивается переменной **res** (от английского result – результат). Инструкция **re.open("GET", res, true)** формирует запрос, где указано, что информация будет передана методом GET по адресу, хранящемуся в переменной **res**. Третий параметр – **true** – сообщает, что запрос должен происходить асинхронно. Наконец метод **re.send()** передает запрос на сервер.

XMLHttpRequest *– если попытаться описать этот объект терминами из программирования, то получится довольно замысловатая характеристика: класс прикладного интерфейса, каждый экземпляр которого предоставляет свойства и методы, позволяющие формировать запрос к серверу и получать данные из ответа. Используя несколько иную терминологию, можно сказать, что* ***XMLHttpRequest*** *– это элемент технологии Ajax, позволяющий нам создавать приложения для обмена данными между браузером и сервером, не задумываясь о механизмах такого обмена.*

Естественно, что нужно оповестить клиента об отправке его сообщения. Для этого свойству **display** текстовой строки с ID **att1** мы присваиваем значение **inline**. Внизу, под формой появляется предупреждение «СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...».

Напоминаю, что в предисловии я уже предупреждал вас: программа обработки данных из формы написана на PHP. На это указывает расширение **.php** файла **record**. Нас не интересует содержимое этой программы. Достаточно знать, что она получает информацию и пересылает обратно некий ответ. Чтобы обработать его, мы используем функцию **show()**:

```
function show()
{
var ans=this.responseText;
if(ans==1)
  {
  document.getElementById("att1").style.display="none";
  document.getElementById("att2").style.display="inline";
  }
```

}

Результат ответа сервера присваиваем переменной **ans** (сокращение от answer – ответ, отклик). Если обработка информации прошла успешно, скрипт **record.php** передает обратно число 1. Получив его, функция выполняет два оператора:

```
document.getElementById("att1").style.display="none";
document.getElementById("att2").style.display="inline";
```

Первая инструкция скрывает надпись «СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...», вторая выводит подтверждение «СООБЩЕНИЕ ДОСТАВЛЕНО».

Вот и все. Как видите, мы обошлись без перезагрузки страницы.

Свойство responseText *предоставляет доступ к ответу сервера. Значение* ***responseText*** *– текстовая строка. В нашем случае сервер передает в качестве подтверждения успешной обработки информации число 1.*

Полученный результат:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Ajax</title>
<style>
#att1 {display: none; color: #CC0000;}
#att2 {display: none; color: #0000CC;}
</style>

<script>
window.onload=function()
{
document.getElementById("bu").onclick=ret;
};

function ret()
{
var re=new XMLHttpRequest();
re.onload=show;
var one=document.getElementById("F1").value;
var two=document.getElementById("F2").value;
var tri=document.getElementById("F3").value;
var fou=document.getElementById("F4").value;
var res="record.php?F1="+one+"&F2="+two+"&F3="+tri+"&F4="+fou;
```

```
re.open("GET", res, true);
re.send();
document.getElementById("att1").style.display="inline";
}

function show()
{
var ans=this.responseText;
if(ans==1)
  {
  document.getElementById("att1").style.display="none";
  document.getElementById("att2").style.display="inline";
  }
}
</script>
</head>

<body>

<form>Ваше имя: <input type="text" name="F1" id="F1"><br>
Телефон: <input type="text" name="F2" id="F2"><br>
E-mail: <input type="text" name="F3" id="F3"><br>
Сообщение: <textarea name="F4" id="F4"></textarea><br>
<input id="bu" type="button" value="Отправить">
<span id="att1"><br>СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...</span>
<span id="att2"><br>СООБЩЕНИЕ ДОСТАВЛЕНО</span></form>

</body>
</html>
```

Наш пример написан максимально просто. При желании программу можно усовершенствовать, добавив сообщение, которое будет выводиться, если скрипт **record.php** обнаружит некорректные данные.

Не раз наблюдал ситуации, когда начинающие программисты с опаской смотрели на технологию Ajax. Мысль о том, что они могут не справиться со сложными методами этой технологии, вызывала у них сомнения: а стоит ли вообще браться за Ajax? Надеюсь, приведенный выше пример наглядно демонстрирует: все очень просто. Необходимо только сделать одну оговорку: чтобы подобные методы отправки форм работали «без сучка и задоринки», требуется разместить на сервере программу обработки данных. Для написания такой программы подойдет специализированный язык, например, PHP, Python, Ruby или Perl. Советую вам изучить любой из них, чтобы стать полноценным разработчиком, способным создавать как клиентские, так и серверные продукты.

Продолжим. В нашем примере была использована достаточно простая форма, содержащая только текстовые поля. Но если бы она имела поле для за-

грузки файла, метод GET оказался бы непригоден. В такой ситуации необходимо отправлять данные методом POST. Для подобного случая Ajax предлагает другой способ (но имеющий много общего с рассмотренным ранее). Этот способ можно назвать универсальным. Он подходит для любых форм. Поэтому для первого примера я не стал создавать отдельную страницу на сайте http://bookjs.ru/, а решил сосредоточить все внимание на втором – универсальном – примере. Его работа демонстрируется на странице http://bookjs.ru/f2.html.

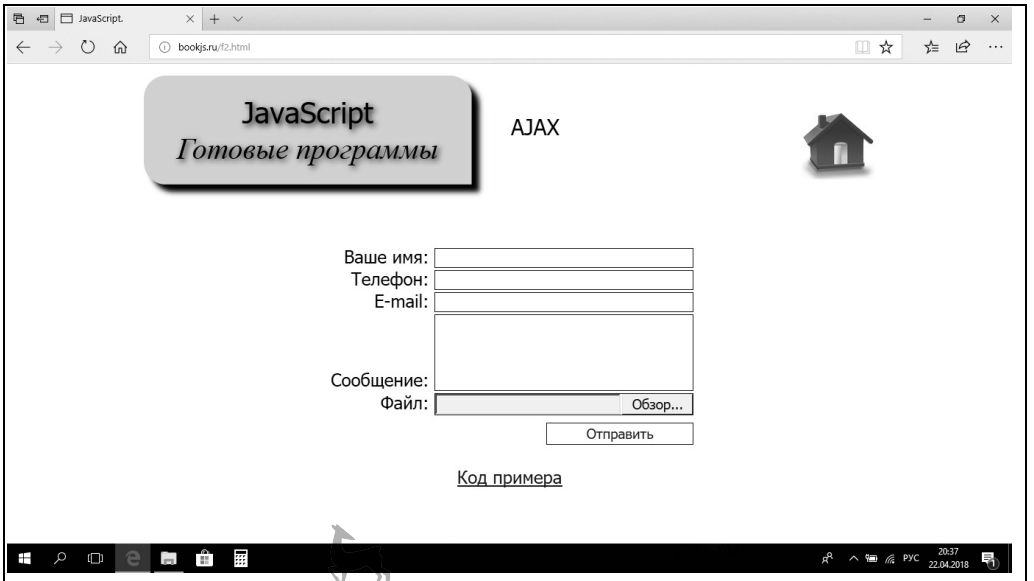

Рис. 3_3. Форма с полем для отправки файла

Изменим форму в соответствии с новой задачей:

```
<form name="shape" id="shape" enctype="multipart/form-data">
Ваше имя: <input type="text" name="F1" id="F1"><br>
Телефон: <input type="text" name="F2" id="F2"><br>
E-mail: <input type="text" name="F3" id="F3"><br>
Сообщение: <textarea name="F4" id="F4"></textarea><br>
Файл: <input type="file" name="F5" id="F5"><br>
<input id="bu" type="button" value="Отправить">
<span id="att1"><br>СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...</span>
<span id="att2"><br>СООБЩЕНИЕ ДОСТАВЛЕНО</span></form>
```

В новой версии форма имеет название **shape** (что так и переводится – форма). Появилось поле загрузки файла **<input type="file" name="F5" id="F5">**. На рисунке 3_3 эти изменения представлены наглядно.

Изменения коснулись и функции **ret()**:

```
function ret()
{
var sh=new FormData(document.forms.shape);
var re=new XMLHttpRequest();
re.onload=show;
re.open("POST", "record.php", true);
re.send(sh);
document.getElementById("att1").style.display="inline";
}
```

Первое, на что сразу обращаешь внимание – исчезли строки, в которых четырем переменным присваивались значения полей формы. Кроме того, отпала необходимость в переменной **res**. Такая «усушка и утруска» функции случилась, потому что мы перешли с метода GET, в котором все данные пересылаются в строке запроса, на метод POST – при его использовании информация передается в теле запроса.

Наиболее важные изменения произошли в самом начале функции: появилась строка

var sh=new FormData(document.forms.shape);

В ней мы создаем новый объект **FormData**, с помощью которого производится кодирование данных из формы **shape**. Полученный результат присваиваем переменной **sh** (сокращение от shape). Далее, как и в первом примере, создаем новый объект **XMLHttpRequest** и сообщаем программе, что при возвращении ответа от сервера должна запускаться функция **show()**. При формировании запроса указываем метод отправки сообщения – POST. И, наконец, отправляем запрос, помесив в его тело кодированные данные:

re.send(sh);

FormData *– еще один случай, когда объект придется характеризовать наукоемкими терминами: прикладной интерфейс, создающий тело запроса, состоящего из нескольких частей. Такими частями могут быть текст или файлы. Удобен для создания Ajax-запроса с данными в формате multipart/form-data. Позволяет отправлять данные из форм, не задумываясь о механизме передачи информации на сервер.*

Последняя строка

document.getElementById("att1").style.display="inline";

как и в предыдущем случае, необходима для вывода предупреждения «СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...».

Функция **show()** никаких изменений не претерпела. Она выполняет свою прежнюю «работу» – выводит подтверждение «СООБЩЕНИЕ ДОСТАВЛЕНО» при успешной обработке данных скриптом **record.php**.

Продемонстрируем полный код второго примера. Вот он:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Ajax</title>
<style>
#att1 {display: none; color: #CC0000;}
#att2 {display: none; color: #0000CC;}
</style>

<script>
window.onload=function()
{
document.getElementById("bu").onclick=ret;
};

function ret()
{
var sh=new FormData(document.forms.shape);
var re=new XMLHttpRequest();
re.onload=show;
re.open("POST", "record.php", true);
re.send(sh);
document.getElementById("att1").style.display="inline";
}

function show()
{
var ans=this.responseText;
if(ans==1)
  {
  document.getElementById("att1").style.display="none";
  document.getElementById("att2").style.display="inline";
  }
}
</script>
</head>

<body>
```

```
<form name="shape" id="shape" enctype="multipart/form-data">
Ваше имя: <input type="text" name="F1" id="F1"><br>
Телефон: <input type="text" name="F2" id="F2"><br>
E-mail: <input type="text" name="F3" id="F3"><br>
Сообщение: <textarea name="F4" id="F4"></textarea><br>
Файл: <input type="file" name="F5" id="F5"><br>
<input id="bu" type="button" value="Отправить">
<span id="att1"><br>СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...</span>
<span id="att2"><br>СООБЩЕНИЕ ДОСТАВЛЕНО</span></form>

</body>
</html>
```

---------- КОД СКРИПТА record.php ----------

```
<?php
echo "1";
```

ВНИМАНИЕ!
Здесь приведен вариант максимально простой серверной программы record.php. В других примерах мы также будем использовать очень простые серверные сценарии, написанные на PHP. Все они годятся ТОЛЬКО в демонстрационных целях. Учтите, что аналоги данных программ, используемых в реальных проектах, гораздо сложнее. В первую очередь, создавая серверные приложения, необходимо продумать меры безопасности, чтобы перекрыть недоброжелателям любые возможности нанести урон вашему сайту.

Если вы пожелаете использовать этот пример в своих проектах, нужно будет только изменить количество полей формы и имя файла, который обрабатывает данные на сервере.

Таймер загрузки

Отправка формы и ответ сервера может занять всего пару секунд. Но иногда на это требуется гораздо больше времени, особенно, когда вы закачиваете большой файл «весом» в несколько мегабайт. Если процесс затягивается, необходимо предупредить клиента, что страница не зависла, закачка идет успешно и надо просто немного подождать.

В примерах данной главы для уведомления о том, что происходит отправка формы, был использован текст «СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...». Это один способ предупреждения. Другой состоит в том, что на странице появляется некое динамически активное изображение, показывающее, что процесс идет. В качестве таких изображений обычно используют линейные или круго-

вые индикаторы. Одни создаются с помощью графических редакторов, другие – программными методами.

Поскольку наша книга посвящена JavaScript, мы попробуем сделать таймер загрузки средствами этого языка. Полученный результат представлен на странице http://bookjs.ru/f3.html.

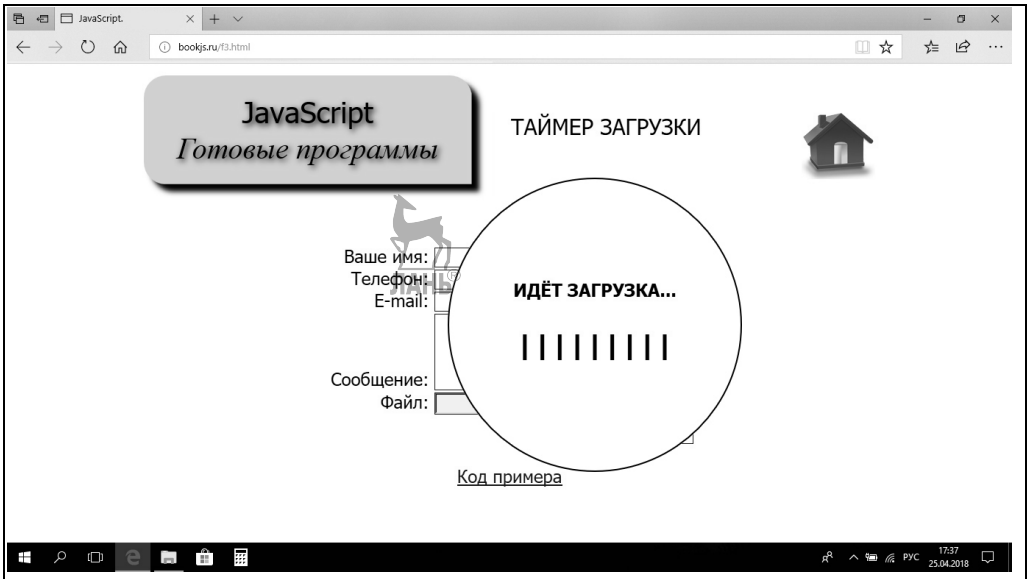

Рис. 3_4. Таймер загрузки данных из формы

Воспользуемся формой из предыдущего раздела:

```
<form name="shape" id="shape" enctype="multipart/form-data">
Ваше имя: <input type="text" name="F1" id="F1"><br>
Телефон: <input type="text" name="F2" id="F2"><br>
E-mail: <input type="text" name="F3" id="F3"><br>
Сообщение: <textarea name="F4" id="F4"></textarea><br>
Файл: <input type="file" name="F5" id="F5"><br>
<input id="bu" type="button" value="Отправить">
<span id="att2"><br>СООБЩЕНИЕ ДОСТАВЛЕНО</span></form>
```

Строка с текстом «СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...» нам уже не нужна, поэтому удалим ее из разметки. Останется только подтверждение успешной отправки формы:

```
<span id="att2"><br>СООБЩЕНИЕ ДОСТАВЛЕНО</span>
```

Теперь надо решить, как будет выглядеть наш индикатор. Из множества возможных вариантов автор выбрал круглую картинку с надписью «ИДЁТ

ЗАГРУЗКА...» и линейным графическим таймером. Внешний вид этой «конструкции» показан на рисунке 3_4.

Чтобы получить такое изображение, необходимо добавить на страницу (в тело документа) следующий код:

```
<table id="tab">
<tr><td><p>ИДЁТ ЗАГРУЗКА...</p>
<p id="loa"></p></td></tr>
</table>
```

В теге head размещаем настройки:

```
#tab {width: 400px; height: 400px; background: #FFFFFF; font-family: Tahoma;
font-size: 24px; font-weight: bold; color: #000000; border: 2px solid #000000;
border-radius: 200px; position: absolute; z-index: 55; visibility: hidden;}
p {text-align: center;}
#loa {font-size: 36px;}
#att2 {display: none; color: #0000CC;}
```

Таблица в качестве базы для индикатора была использована с целью упростить позиционирование элементов. Внешний вид надписи, а также «палочек» таймера определяются свойствами

```
font-family: Tahoma; font-weight: bold; color: #000000;
```

Для свойства **border-radius** установлено значение **200px** – это придает индикатору форму круга. В исходном состоянии индикатор не виден (**visibility: hidden**). Значение **z-index** выбрано произвольно.

Строка

```
<p id="loa"></p>
```

предназначена для вывода показаний таймера. В начальном состоянии она пустая.

Разберем принцип работы нашей «системы».

Посетитель заполнил форму и нажал кнопку «Отправить», приведя тем самым в действие функцию **ret()**:

```
function ret()
{
var sh=new FormData(document.forms.shape);
var re=new XMLHttpRequest();
re.onload=show;
re.open("POST", "record.php", true);
```

```
re.send(sh);
race();
}
```

Внутри функции **ret()** все «по-старому», за исключением последней строки. Раньше она содержала инструкцию показать уведомление «СООБЩЕНИЕ ОТПРАВЛЯЕТСЯ...». Теперь вместо этого в последней строке – вызов новой функции **race()** (термин переводится как бег – подходящее название, чтобы описать поведение таймера). Функция **race()** стартует сразу после отправки данных из формы и необходима для управления показаниями таймера загрузки.

Перед описанием функции **race()** мы инициировали три переменные: выключатель таймера **k**, счетчик **i** и массив **d**:

```
var k=0;
var i=0;
var d=new Array(" ", "|", "|||", "|||||", "|||||||", "|||||||||",
"|||||||||||", "|||||||||||||", "|||||||||||", "|||||||||", "|||||||", "|||||", "|||", "|");
```

Переменная **k** необходима для определения момента остановки таймера. Пока **k==0**, функция **race()** работает.

Счетчик **i** и массив **d** находятся в тесном взаимодействии – это видно из описания функции:

```
function race()
{
if(k==0)
  {
  document.getElementById("tab").style.visibility='visible';
  document.getElementById("loa").innerHTML=d[i];
  i++;
  if(i==13)
    {
    i=0;
    }
  window.setTimeout(race, 200);
  }
}
```

Первое, что делает функция – показывает индикатор **tab** (от английского tablo) на странице:

```
document.getElementById("tab").style.visibility='visible';
```

Инструкция **window.setTimeout(race, 200)** создает замкнутый цикл. Каждые 200 миллисекунд значение счетчика увеличивается на единицу и в строке

показаний таймера (**id="loa"**) выводится элемент массива **d[i]**. В начальном состоянии значение счетчика установлено 0. Поэтому таймер демонстрирует только надпись «ИДЁТ ЗАГРУЗКА...», так как первый элемент массива – пробел. Спрашивается, почему **" "**, а не просто **""**, ведь при обоих вариантах строка таймера будет пустой? Если не использовать пробел, в первый момент уведомление «ИДЁТ ЗАГРУЗКА...» окажется по вертикали строго по центру, так как у нижней строки нет содержимого. А затем, когда в строке таймера появится первый элемент, уведомление сместится выше от центра. Выглядит это как «прыжок» текста вверх. **d[0]=" "** избавляет нас от таких «прыжков».

Идем дальше. Когда переменная **i** получает значение 1, в строке таймера появляется изображение |, когда **i** увеличивается еще на единицу, таймер показывает |||. И так далее. Выглядит это так, словно линии сначала разбегаются в стороны от центра, а потом возвращаются обратно. Наконец, показан последний, 13-й элемент массива. Счетчик **i** обнуляется и процесс начинается заново.

Так происходит до тех пор, пока не будет получен ответ сервера. Наконец ответ пришел. В этот момент в дело вступает функция **show()**:


```
function show()
{
var ans=this.responseText;
if(ans==1)
  {
  k=1;
  document.getElementById("tab").style.visibility="hidden";
  document.getElementById("att2").style.display="inline";
  }
}
```


Первым делом переменной **k** присваивается значение 1 и функция **race()** останавливается. Следующая инструкция скрывает индикатор загрузки. Последняя инструкция меняет значение свойства **display** строки подтверждения и она появляется на экране компьютера.

Собрав все компоненты воедино, мы получим такую страницу:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Таймер загрузки</title>
<style>
#tab {width: 400px; height: 400px; background: #FFFFFF; font-family: Tahoma;
font-size: 24px; font-weight: bold; color: #000000; border: 2px solid #000000;
border-radius: 200px; position: absolute; z-index: 55; visibility: hidden;}
```

```
p {text-align: center;}
#loa {font-size: 36px;}
#att2 {display: none; color: #0000CC;}
</style>

<script>
window.onload=function()
{
document.getElementById("bu").onclick=ret;
};

function ret()
{
var sh=new FormData(document.forms.shape);
var re=new XMLHttpRequest();
re.onload=show;
re.open("POST", "record.php", true);
re.send(sh);
race();
}

var k=0;
var i=0;
var d=new Array(" ", "|", "|||", "|||||", "|||||||", "|||||||||",
"|||||||||||", "|||||||||||||", "|||||||||||", "|||||||||", "|||||||", "|||||", "|||", "|");

function race()
{
if(k==0)
  {
  document.getElementById("tab").style.visibility='visible';
  document.getElementById("loa").innerHTML=d[i];
  i++;
  if(i==13)
    {
    i=0;
    }
  window.setTimeout(race, 200);
  }
}

function show()
{
var ans=this.responseText;
if(ans==1)
```

```
  {
  k=1;
  document.getElementById("tab").style.visibility="hidden";
  document.getElementById("att2").style.display="inline";
  }
}
</script>
</head>

<body>

<table id="tab">
<tr><td><p>ИДЁТ ЗАГРУЗКА...</p>
<p id="loa"></p></td></tr>
</table>

<form name="shape" id="shape" enctype="multipart/form-data">
Ваше имя: <input type="text" name="F1" id="F1"><br>
Телефон: <input type="text" name="F2" id="F2"><br>
E-mail: <input type="text" name="F3" id="F3"><br>
Сообщение: <textarea name="F4" id="F4"></textarea><br>
Файл: <input type="file" name="F5" id="F5"><br>
<input id="bu" type="button" value="Отправить">
<span id="att2"><br>СООБЩЕНИЕ ДОСТАВЛЕНО</span></form>

</body>
</html>
```

Рассмотренный пример имеет элемент незавершенности. Дело в том, что в коде отсутствует позиционирование индикатора на странице. Так сделано для упрощения кода (читатель, наверное, уже обратил внимание, что этот мотив играет решающую роль при составлении программ для данной книги). Если вы планируете использовать этот или подобный вариант таймера в своих проектах, необходимо будет позаботиться о том, чтобы он появлялся рядом с местом щелчка на кнопке «Отправить». Для этих целей вы можете воспользоваться методом позиционирования, который мы разбирали во второй главе, когда рассказывали о выпадающих меню.

При создании эффекта разбегающихся и возвращающихся линий автор использовал прямолинейный подход (опять же для простоты кода) – создал массив, каждый элемент которого содержал необходимое количество линий. Но это не единственно доступный вариант. Можно вообще избавиться от массива и создать цикл, в котором на каждом витке нужное число «палочек» создавалось бы «на лету». Конечно, такой вариант будет сложнее, зато любители экспериментов смогут попрактиковаться в усовершенствовании готовых программ.

Подсказки

Как утверждает народная мудрость, никто не застрахован от ошибок. В том числе и посетители вашего сайта, заполняющие форму отправки сообщения. Часто ошибки возникают, когда клиент не знает, в каком виде необходимо ввести информацию. Например, вы управляете интернет-магазином. На странице оформления заказа есть строка для телефонного номера. Получив его, серверная программа отправляет клиенту СМС с подтверждением заявки. Допустим, вы ожидаете поступления номера, в котором будут одни цифры: 89037500877. А получаете 8-903-750-08-77. Программа обработки данных не распознает номер и уведомление не будет отправлено.

Во избежание подобных ошибок дальновидные разработчики в соответствующих полях формы демонстрируют примеры правильного заполнения – они служат подсказками для клиентов.

Другой случай. Если вы пользуетесь социальными сетями, то не раз оставляли сообщения на своих страницах или в группах. В текст сообщения можно добавить адрес какого-нибудь сайта (и даже не одного). Когда ваша информация появляется на странице, адрес сайта преобразуется в ссылку (иногда еще говорят, что ссылка в тексте «подсвечивается»). В свое время автор провел сравнительный анализ механизма «подсвечивания» в двух популярных сетях: Фейсбук и ВКонтакте. И обнаружил, что в Фейсбуке программа определения ссылок написана настолько добросовестно, что даже некорректно введенные адреса сайтов распознаются правильно. А вот ВКонтакте не может похвастаться такими результатами, поэтому не мешало бы его разработчикам добавить комментарии по правильному оформлению ссылок.

Думаю, двух примеров достаточно. Настало время поговорить о создании подсказок. В принципе, это очень простое занятие – ввел в поле необходимый образец – и подсказка готова. Осталось научиться убирать ее, как только клиент начнет заполнять соответствующее поле.

Автор в качестве образца выбрал случай с подсказкой по правильному вводу адреса сайта. Страница с этим примером – http://bookjs.ru/f4.html.

Поскольку принцип создания и скрытия подсказок не зависит от количества полей, мы оставим в форме только textarea (рис. 3_5).

Разместим в поле текст, который будет показываться после загрузки страницы в браузере:

```
"Если вы хотите ввести адрес ссылки,
начинайте его с www, http:// или https://"
```

Обратите внимание – мы разделили текст на строки, чтобы сделать акцент на его отдельные части.

Разметка выглядит так:

```
<form>Сообщение:
<textarea name="F4" id="F4">Если вы хотите ввести адрес ссылки,
начинайте его с www, http:// или https://</textarea></form>
```

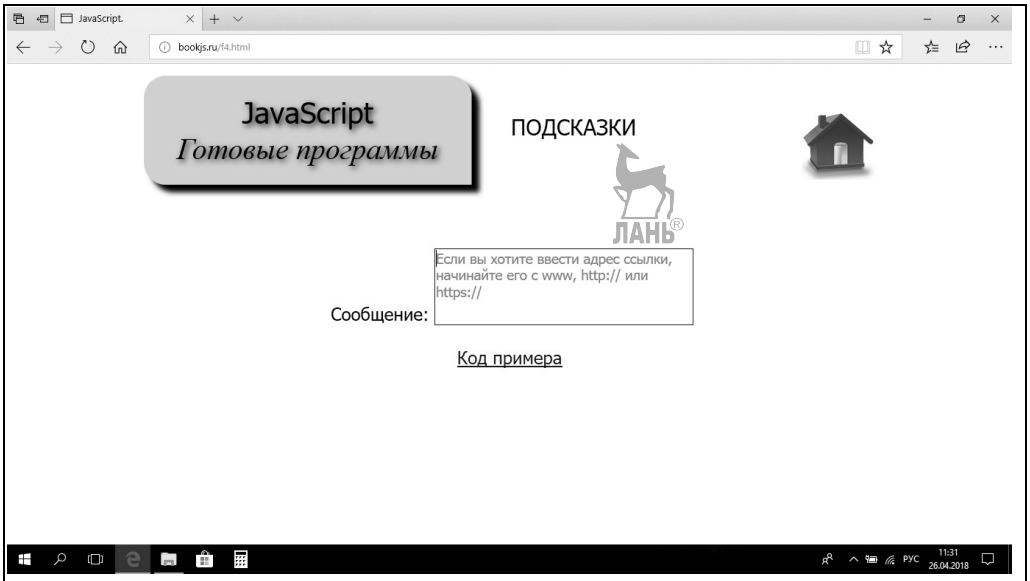

Рис. 3_5. Текстовое поле с подсказкой

Здесь важную роль играют параметры стиля:

textarea {font-family: Tahoma; font-size: 18px;
color: #808080; width: 350px; height: 100px;}

Мы выбрали шрифт и сделали его бледно-серым. Это нужно для контраста с черным шрифтом, которым отображается текст в поле при его наборе клиентом.

Поскольку наша задача продемонстрировать метод удаления подсказки из текстового поля, а не отправлять форму (это мы уже умеем), мы не стали давать форме наименование и исключили функции ее отправки. Но добавили функцию **tip()** (переводится как подсказка). Вызов функции происходит при щелчке мышью в поле **textarea**:

window.onload=function()
{
document.getElementById("F4").onclick=tip;
};

Новая функция выполняет проверку: какой это клик по счету. Если первый, текст подсказки удаляется из поля. На все последующие щелчки функция не реагирует.

Начнем с того, что инициируем переменную **f**, которую будем использовать в качестве идентификатора начального состояния:

```
var f=0;
```

При первом клике на текстовом поле выполняется первый вызов функции:

```
function tip()
{
if(f==0)
  {
  document.getElementById("F4").value="";
  document.getElementById("F4").style.color="#000000";
  f=1;
  }
}
```

И первым делом (прошу прощения за тавтологию) проверяется условие **f==0**. Если оно верно, текст подсказки удаляется из поля:

```
document.getElementById("F4").value="";
```

Свойству **color** шрифта текстового поля присваивается значение **#000000**. Теперь любые введенные в поле символы будут отображаться черным цветом. Последний оператор присваивает переменной **f** значение 1. Отныне любой щелчок в **textarea** не вызовет никакой реакции, так как условие **f==0** будет ложным.

В итоге у нас получилась одна из самых простых программ в этой книге:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Подсказки</title>
<style>
textarea {font-family: Tahoma; font-size: 18px;
color: #808080; width: 350px; height: 100px;}
</style>

<script>
window.onload=function()
{
document.getElementById("F4").onclick=tip;
};

var f=0;

function tip()
```

```
{
if(f==0)
  {
  document.getElementById("F4").value="";
  document.getElementById("F4").style.color="#000000";
  f=1;
  }
}
</script>
</head>

<body>

<form>
Сообщение: <textarea name="F4" id="F4">Если вы хотите ввести адрес
ссылки,
начинайте его с www, http:// или https://</textarea></form>

</body>
</html>
```

Проверка формы

Если вам уже приходилось читать о работе с формами, вы могли заметить, что во всех случаях авторы обязательно заводят речь о проверке информации, поступающей от посетителей. А они четко делятся на две неравные группы: добросовестные клиенты (их подавляющее большинство) и злоумышленники (их меньшинство).

Как и в предыдущем разделе, вновь констатируем факт: никто не застрахован от ошибок. Из-за спешки или невнимательности посетители могут отправить вам не те данные, что вы ожидаете. Или вообще случайно пропустить и оставить незаполненным какое-то поле. Другой пример непреднамеренной ошибки – отправка файла непредусмотренного формата. Вы ждете JPG, PNG или GIF, а получаете BMP, с которым ваш сайт не работает.

Добросовестным клиентам обязательно надо помочь. Для этого разработчики используют не только подсказки, но и проверку формы перед ее отправкой.

Совсем другая история с недоброжелателями, которые пытаются навредить вашему сайту. Классический пример, описанный уже неоднократно: некто создает на своем компьютере страницу с такой же формой, как на вашем сайте, но без проверки данных. Затем помещает в форму вредоносное содержимое и отправляет его. Если серверная программа обработки данных не имеет защиты от такого содержимого – ждите проблем. Поэтому всю входящую информацию необходимо проверять не только на странице сайта, но и на сервере. Причем

контроль на стороне сервера должен быть более строгим, ведь именно здесь ставится барьер для недоброжелателей. Надо обязательно проверять объем входящей информации, типы файлов, наличие исполняемого кода и запрещенных символов, соответствие ссылок требуемому формату и разрешенным адресам.

В книге мы ведем разговор о программах, работающих на стороне клиента, поэтому наша обязанность – помочь добросовестным посетителям отправить корректно заполненную форму. Этим мы и займемся. А поскольку разобрать все возможные (и весьма многочисленные) варианты проверок – слишком глобальная задача, автор выбрал простой пример, на основе которого можно наглядно продемонстрировать подход к решению подобных задач. Посмотреть, как работает проверка небольшой формы, можно на странице http://bookjs.ru/f5.html.

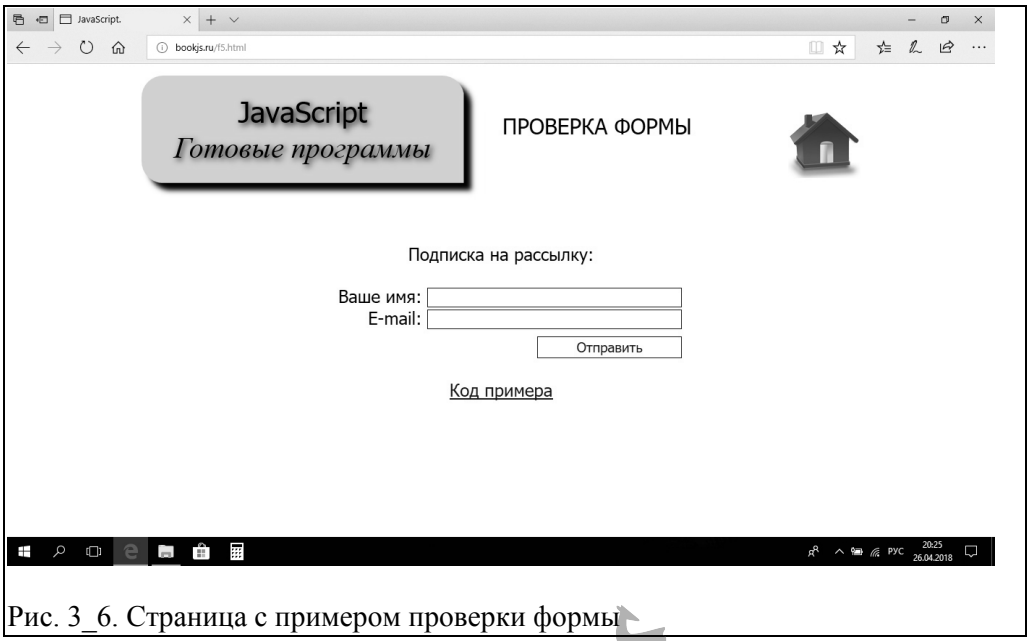

Рис. 3_6. Страница с примером проверки формы

Предположим, что на нашем сайте посетители могут подписаться на рассылку новостей или уведомлений. Создадим форму подписки из двух полей: «Ваше имя» и «E-mail» (рис. 3_6):

```
<form name="shape" id="shape">
Ваше имя: <input type="text" name="F1" id="F1"><br>
E-mail: <input type="text" name="F2" id="F2"><br>
<span id="fo"></span>
<input id="bu" type="button" value="Отправить"></form>
```

Строка

```
<span id="fo"></span>
```

необходима для вывода предупреждений в процессе проверки формы. Предупреждения об ошибках показываются красным цветом:

```
#fo {display: none; color: #CC0000;}
```

Для отправки данных снова используем функцию **ret()**. Правда, есть одно «но». До сих пор функция имела изъян: она выполнялась независимо от того, ввел ли посетитель какие-то данные в форме или нет. Теперь мы исправим этот недочет и добавим проверку для каждого текстового поля.

Сначала убедимся, что посетитель не оставил строку «Ваше имя» пустой:

```
if(document.getElementById("F1").value.length<2)
  {
  …
  }
```

Свойство length *возвращает количество знаков в любой рассмотренной строке (условно назовем ее string):* ***var len= string.length****.*

Самые короткие имена состоят из двух букв (например, Ия или Ян). Значит, наличие в строке «Ваше имя» меньшего количества символов является ошибкой, о чем мы и предупредим клиента:

```
document.getElementById("fo").innerHTML="<br>ВЫ НЕ ВВЕЛИ
ИМЯ !<br>";
document.getElementById("fo").style.display="inline";
```

Свойство innerHTML *возвращает HTML-код содержимого элемента. Используя это свойство, можно не только считывать информацию из элемента, но и менять ее или удалять.*

Строку «E-mail» необходимо проверить более тщательно. Самый короткий адрес электронной почты обязан содержать знак @ («собака» или коммерческое «эт»). Кроме того, должна присутствовать точка, разделяющая доменные имена первого и второго уровня. Она будет отстоять от первого символа имени как минимум на пять позиций (два символа в начале + @ + два символа доменного имени второго уровня). Также адрес должен иметь длину не менее 8 символов: два в начале + @ + два символа доменного имени второго уровня + точка + два символа доменного имени первого уровня (теоретически возможны и более короткие имена, но они настолько уникальное явление, что учитывать их существование не имеет смысла). Проверяем:

```
var d=document.getElementById("F2").value;
if((d.length<8)||(d.indexOf("@")<2)||(d.indexOf(".")<5))
  {
```

```
  document.getElementById("fo").innerHTML="<br>ВЫ НЕ ВВЕЛИ E-
MAIL !<br>";
  document.getElementById("fo").style.display="inline";
  }
```

Если оба поля заполнены верно, отправляем форму и добавляем в конце функции уведомление:

```
document.getElementById("fo").innerHTML="<br>ДАННЫЕ
ОТПРАВЛЯЮТСЯ...<br>";
document.getElementById("fo").style.display="inline";
```

Получилась новая версия функции **ret()**:

```
function ret()
{
if(document.getElementById("F1").value.length<2)
  {
  document.getElementById("fo").innerHTML="<br>ВЫ НЕ ВВЕЛИ
ИМЯ !<br>";
  document.getElementById("fo").style.display="inline";
  }
else
  {
  var d=document.getElementById("F2").value;
  if((d.length<8)||(d.indexOf("@")<2)||(d.indexOf(".")<5))
    {
    document.getElementById("fo").innerHTML="<br>ВЫ НЕ ВВЕЛИ E-
MAIL !<br>";
    document.getElementById("fo").style.display="inline";
    }
  else
    {
    var sh=new FormData(document.forms.shape);
    var re=new XMLHttpRequest();
    re.onload=show;
    re.open("POST", "record.php", true);
    re.send(sh);
    document.getElementById("fo").innerHTML="<br>ДАННЫЕ
ОТПРАВЛЯЮТСЯ...<br>";
    document.getElementById("fo").style.display="inline";
    }
  }
}
```

Проверка выполняется последовательно. Пока поле «Ваше имя» не заполнено, любое нажатие кнопки «Отправить» вызывает только демонстрацию предупреждения «ВЫ НЕ ВВЕЛИ ИМЯ!» (рис. 3_7). Лишь после ввода имени функция переходит к проверке второго условия. Оно состоит из трех частей. Если не выполнены все (недостаточное количество символов, отсутствие знака @, отсутствие точки) или хотя бы одно из них – демонстрируется предупреждение «ВЫ НЕ ВВЕЛИ E-MAIL!». И опять: сколько не нажимай кнопку, отправка формы не происходит. Это случится, когда второе поле тоже будет заполнено правильно. Вот теперь можно жать на кнопку!

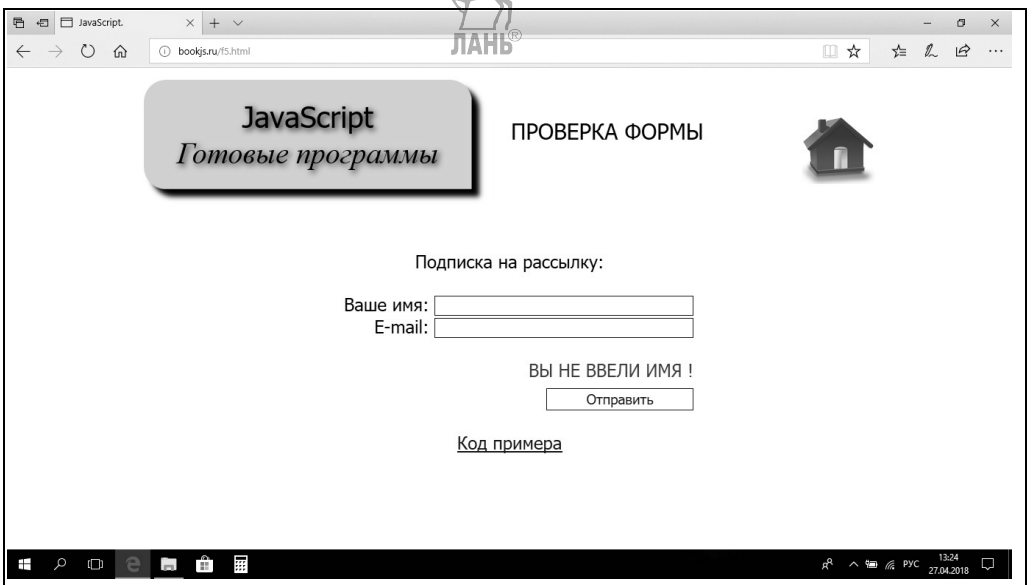

Рис. 3_7. Если посетитель оставил незаполненным одно или оба поля, появляются соответствующие предупреждения

Как и в предыдущих случаях, ответ сервера обрабатывает функция **show()**:

```
function show()
{
var ans=this.responseText;
if(ans==1)
  {
  document.getElementById("fo").innerHTML="<br>ВЫ ПОДПИСАНЫ НА
УВЕДОМЛЕНИЯ<br>";
  document.getElementById("fo").style.color="#0000CC";
  document.getElementById("fo").style.display="inline";
  }
}
```

Если информация передана успешно, появляется сообщение «ВЫ ПОДПИСАНЫ НА УВЕДОМЛЕНИЯ». Инструкция

```
document.getElementById("fo").style.color="#0000CC";
```

меняет цвет шрифта, поэтому текст подтверждения будет синим.

Как всегда, страница целиком:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Проверка формы</title>
<style>
#fo {display: none; color: #CC0000;}
</style>

<script>
window.onload=function()
{
document.getElementById("bu").onclick=ret;
};

function ret()
{
if(document.getElementById("F1").value.length<2)
  {
  document.getElementById("fo").innerHTML="<br>ВЫ НЕ ВВЕЛИ
ИМЯ !<br>";
  document.getElementById("fo").style.display="inline";
  }
else
  {
  var d=document.getElementById("F2").value;
  if((d.length<8)||(d.indexOf("@")<2)||(d.indexOf(".")<5))
    {
    document.getElementById("fo").innerHTML="<br>ВЫ НЕ ВВЕЛИ E-
MAIL !<br>";
    document.getElementById("fo").style.display="inline";
    }
  else
    {
    var sh=new FormData(document.forms.shape);
    var re=new XMLHttpRequest();
    re.onload=show;
```

```
    re.open("POST", "record.php", true);
    re.send(sh);
    document.getElementById("fo").innerHTML="<br>ДАННЫЕ
ОТПРАВЛЯЮТСЯ...<br>";
    document.getElementById("fo").style.display="inline";
    }
  }
}

function show()
{
var ans=this.responseText;
if(ans==1)
  {
  document.getElementById("fo").innerHTML="<br>ВЫ ПОДПИСАНЫ НА
УВЕДОМЛЕНИЯ<br>";
  document.getElementById("fo").style.color="#0000CC";
  document.getElementById("fo").style.display="inline";
  }
}
</script>
</head>

<body>

<form name="shape" id="shape">
Ваше имя: <input type="text" name="F1" id="F1"><br>
E-mail: <input type="text" name="F2" id="F2"><br>
<span id="fo"></span>
<input id="bu" type="button" value="Отправить"></form>

</body>
</html>
```

Хочу обратить внимание на проверку наличия точки в адресе почты. На самом деле, эта проверка не гарантирует, что e-mail будет указан правильно. Представьте себе такую ситуацию. У клиента есть почта – pupkin.v@abvgd.ru. Если он допустит ошибку и введет pupkin.v@abvgdru, программа посчитает адрес корректным, так как точка стоит на седьмой позиции, и «не заметит», что доменные имена не разделены. Поэтому проверка точки эффективна только в тех адресах, где она в единственном числе или стоит после @.

Для контроля заполнения формы мы использовали простые методы: определяли длину введенного текста и наличие в нем необходимых символов. Но посетитель может выполнить эти требования и одновременно допустить ошибку, которая исказит информацию. Например, на моем компьютере буква «э» со-

вмещена со знаком дефиса. Для печати буквы необходимо нажать клавишу «Fn». Если в спешке забыть об этом, то вместо имени Эдуард можно напечатать –дуард. Конечно, администратор сайта, скорее всего, правильно интерпретирует имя. Хуже, когда клиент случайно введет не тот символ в адресе электронной почты. Опять же, в качестве примера возьму свой компьютер. У меня на клавиатуре рядом с «z» клавиша «\». Промахнулся – и в адресе почты серьезная ошибка, которая не позволит администратору распознать истинный e-mail (на разных клавиатурах расположение клавиш отличается друг от друга – попробуй угадай, мимо какой промазал посетитель).

Поэтому для максимальной минимизации ошибок (хорошая фраза – максимальная минимизация) проверку нужно выполнять более тщательно. Используя регулярные выражения, можно создать ряд дополнительных ограничений для клиента:

– разрешить в поле «Ваше имя» использовать только буквы русского или английского алфавита;

– разрешить в поле «E-mail» использовать только точку, дефис, нижнее подчеркивание, @, латинские буквы и арабские цифры (вообще-то стандартами разрешено использовать и некоторые другие символы, но популярные почтовые серверы не разрешают пользователям создавать логины с такими символами).

Читатели могут самостоятельно внести необходимые дополнения в программу. Перед вами большой простор для совершенствования навыков программирования.

Глава 4.
Редакторы для CMS

Я занимаюсь программированием давно. За эти годы прочел много компьютерной литературы, в том числе по JavaScript. Конечно, мне не довелось просмотреть все книги по этому языку, что вышли, допустим, в последнее десятилетие. Но, основываясь на изучении тех изданий, что попадались мне в руки, рискну поделиться одним наблюдением: очень мало авторов касаются темы создания редакторов контента для различных CMS. А ведь это одна из важнейших областей применения JavaScript.

Аббревиатура CMS происходит от английского Content management system, что переводится как система управления содержимым. Многие современные сайты (если не большинство) управляются такими системами. CMS позволяют администратору вносить любые изменения на страницы сайта непосредственно через браузер. Для этого системы управления снабжают визуальными или HTML-редакторами (а часто и теми, и другими).

В этой главе мы с вами попробуем восполнить недостаток информации по редакторам контента. Давайте вместе пройдем, в общем-то, не очень сложный путь – от простейшего редактора до WYSIWYG с функцией предпросмотра обновленной страницы.

Простейший редактор

Представим такую ситуацию: вы разработали сайт, состоящий из нескольких HTML-страниц, и разместили его в сети. Пройдет немного времени – и вы обнаружите, что информацию на одной или нескольких страницах необходимо отредактировать, а может и кардинально изменить. Даже самые стабильные интернет-ресурсы нуждаются в периодическом обновлении. А что говорить про сайты, которые рассказывают о быстро меняющихся вещах, например, о строительстве нового микрорайона или о новинках в мире компьютеров. В таких случаях обновление информации происходит чуть ли не каждый день.

И вот вы стоите перед задачей ежедневно редактировать одну или несколько страниц и ставить их вместо устаревших версий. Выглядит это примерно так: открыли на своем компьютере папку с файлами сайта, добавили новые рисунки, схемы или фотографии, отредактировали страницу, подключились к хостингу, закачали новые изображения, заменили страницу, запустили браузер, проверили полученный результат. Если что-то не понравилось, вновь правите страницу и закачиваете ее новую версию на хостинг. Уже чувствуется, что это довольно хлопотное занятие. Между тем, если бы ваш сайт был снабжен CMS, все оказалось бы гораздо проще: запустили в браузере систему управления и выполняете все манипуляции через нее. Не надо править страницу на своем компьютере, не надо подключаться к хостингу, не надо закачивать отредактированную страницу через файловый менеджер. Еще раз повторюсь: все делается через браузер.

К сожалению, рассказ о таком замечательном программном обеспечении, как CMS, не входит в наши планы. Эти системы пишутся на специальных языках программирования, таких как PHP, Python или Perl. Но некоторые заготовки для CMS мы можем сделать на JavaScript. Речь, как мы говорили выше, идет о редакторах контента.

Как выглядит типичная страница сайта? У нее есть «шапка», нижний колонтитул, меню, блок рекламы и – самое главное – информационный блок. Структура такой среднестатистической страницы показана на рисунке 4_1.

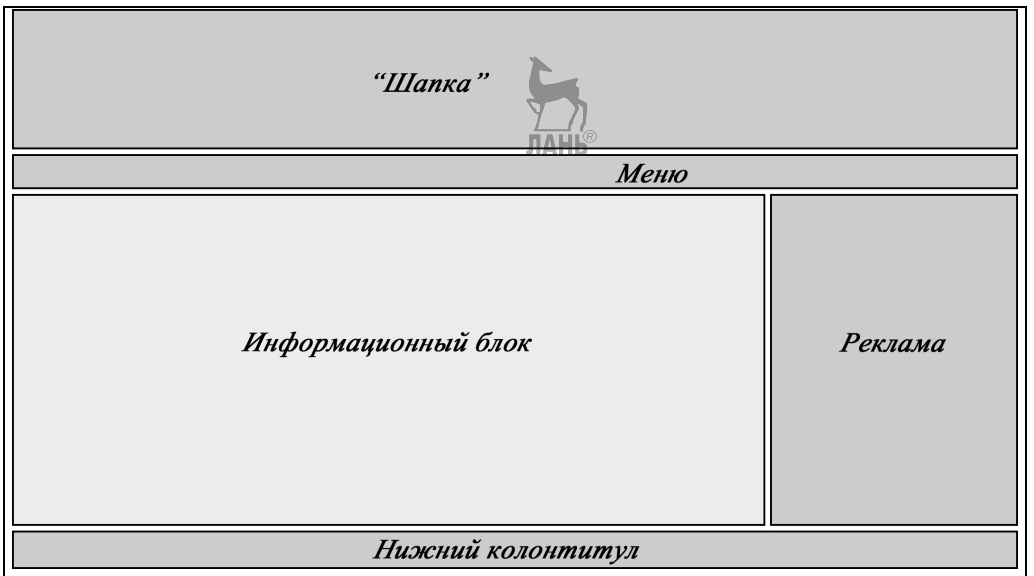

Рис. 4_1. Схема расположения элементов страницы

Самый динамичный, наиболее часто меняемый блок – информационный. Для него мы и напишем редактор. Пока – самый простой.

Итак, представим, что у нас есть некая CMS, управляющая сайтом. В одном из ее разделов находится список страниц. Каждый пункт списка – ссылка, предназначенная для загрузки в редактор информационного блока той или иной страницы. Нажмем одну из ссылок. А чтобы понять, что будет дальше, из воображаемой ситуации переместимся в реальность: на страницу http://bookjs.ru/w1.html демонстрационного сайта (рис. 4_2).

Мы оказались на странице простейшего редактора. Подчеркну: этот пример создан просто для демонстрации принципа работы редакторов, поэтому он максимально прост. Наличие текста создает видимость, что информационный блок уже имеет содержание. В коде примера этот текст отсутствует – так бывает в ситуации с новой, только что созданной страницей, которой еще предстоит обзавестись содержанием. В ручном режиме вы можете добавлять в текст любые теги. В автоматическом вам доступны только две функции: выделение час-

ти или целого текста жирным и вставка на страницу изображения планшетного компьютера.

Поэкспериментируем. Выделите пару слов и нажмите кнопку «BOLD». Вы увидите, что эти слова окажутся заключенными в теги ** **. Установите курсор в любой части текста и нажмите кнопку «IMG». В этом месте появится HTML-код **<img src="pict/comp.jpg" alt="Фото">**. Если бы у нашего редактора была кнопка «Записать» (она появится в следующем примере), мы могли бы отправить содержимое специальной программе, которая произвела бы запись в файл или базу данных. В результате у нашей воображаемой страницы произошли бы изменения в информационном блоке.

Код такого редактора довольно прост – под стать простому примеру.

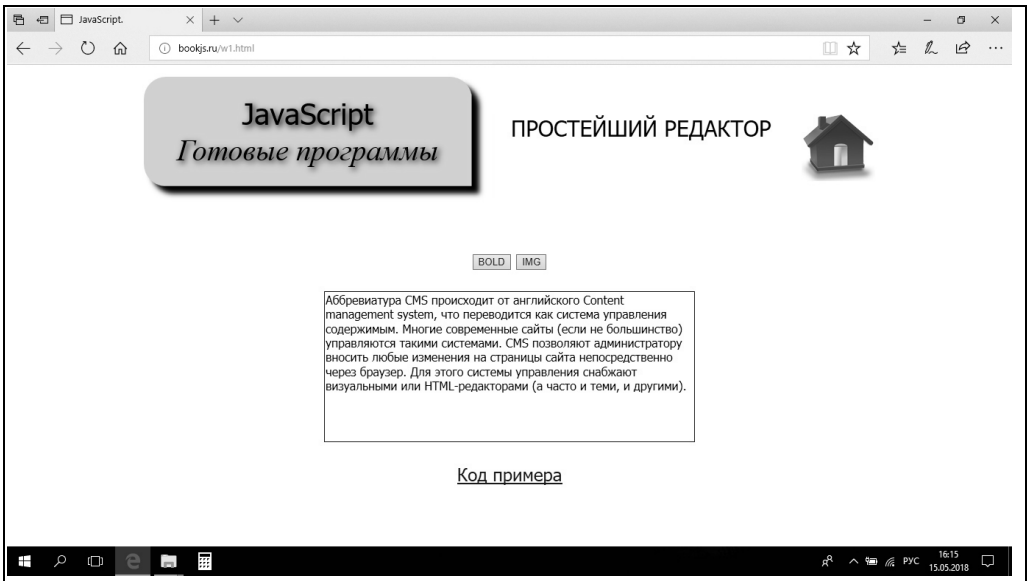

Рис. 4_2. Страница простейшего редактора

Начнем с поля, в которое выводится содержимое информационного блока. Мы воспользуемся элементом <textarea>:

```
<textarea id="tx"></textarea>
```

Над полем разместим две кнопки:

```
<input id="bo" type="button" value="BOLD">
<input id="im" type="button" value="IMG">
<br><br>
<textarea id="tx"></textarea>
```

Настроим параметры текстового поля:

```
<style>
textarea {width: 500px; height: 200px; border: 1px solid #0000CC;
font-family: Tahoma; font-size: 16px; color: #000000;}
</style>
```

Запишем, какие функции должны работать при нажатии на ту или иную кнопку:

```
window.onload=function()
{
document.getElementById("bo").onclick=ins;
document.getElementById("im").onclick=pas;
};
```

Название первой функции происходит от insert – вставить, второй от слова paste – тоже обозначающего вставку.

Выясним, каким образом в тексте появляется жирное начертание слов.

Вы выделили текст и нажали кнопку «BOLD». Первым делом функция **ins()** должна определить выделенный фрагмент:

```
var sest=document.getElementById("tx").selectionStart;
var seen=document.getElementById("tx").selectionEnd;
```

В переменную **sest** помещаем «точку», соответствующую началу выделения, в переменную **seen** позицию, следующую за последним выделенным символом.

Свойство selectionStart *возвращает позицию первого символа в выделенном фрагменте текста. Свойство* ***selectionEnd*** *возвращает позицию первого символа, следующего за выделенным фрагментом – во избежание ошибок обратите внимание на этот нюанс!*

Считываем содержимое текстового поля:

```
var v=document.getElementById("tx").value;
```

Теперь создадим переменную **t**, в которую будет помещен выделенный фрагмент (его мы «вычислим» с помощью метода **substring**):

```
var t=v.substring(sest, seen);
```

Метод substring(a, b) *позволяет выделить последовательную группу символов из строки (часто еще говорят – извлечь подстроку). Символы извлекаются, начиная с позиции* ***a*** *и до позиции, предшествующей* ***b****. Символ из позиции* ***b*** *в подстроку не включается!*

К выделенному тексту надо добавить теги:

''+t+''

Затем считываем текст от начала до исходной точки выделения

v.substring(0, sest)

и от символа, следующего за позицией, обозначающей завершение выделения

v.substring(seen)

Поскольку второй параметр в методе **substring(seen)** отсутствует, строка считывается до конца.

Осталось сформировать новую версию текста

v.substring(0, sest)+''+t+''+v.substring(seen)

и поместить ее в поле **textarea**:

```
document.getElementById("tx").value=v.substring(0,
sest)+'<b>'+t+'</b>'+v.substring(seen);
```

В результате этих «манипуляций» к выделенному фрагменту добавились теги ** **.

Соберем все фрагменты в единое целое:

```
function ins()
{
var v=document.getElementById("tx").value;
var sest=document.getElementById("tx").selectionStart;
var seen=document.getElementById("tx").selectionEnd;
var t=v.substring(sest, seen);
document.getElementById("tx").value=v.substring(0,
sest)+'<b>'+t+'</b>'+v.substring(seen);
}
```

Функция **pas()** еще проще:

```
function pas()
{
var v=document.getElementById("tx").value;
var sest=document.getElementById("tx").selectionStart;
var seen=document.getElementById("tx").selectionEnd;
document.getElementById("tx").value=v.substring(0, sest)+'<img
src="pict/comp.jpg" alt="Фото">'+v.substring(seen);
}
```

Первые три оператора решают те же задачи, что и в функции **ins()**. Последняя строка выполняет вставку изображения в позицию курсора:

```
document.getElementById("tx").value=v.substring(0, sest)+'<
src="pict/comp.jpg">'+v.substring(seen);
```

Если вы не просто установили курсор, а выделили несколько символов, то код изображения появится на их месте, а сами символы исчезнут (или, говоря по-другому, выделенный фрагмент будет заменен рисунком).

Мы разобрали пример элементарного HTML-редактора. Следуя описанной методике, его можно усложнить:

– к уже имеющейся функции выделения жирным добавить другие кнопки форматирования;

– сделать выпадающий список с перечнем всех изображений, имеющихся в «загашнике» сайта;

– создать функции добавления других элементов – таблиц, фреймов, ссылок.

Так что творческие просторы для энтузиастов необъятны. Было бы желание.

Завершая описание нашего первого редактора, остается привести его полный код:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Простейший редактор</title>
<style>
textarea {width: 500px; height: 200px; border: 1px solid #0000CC;
font-family: Tahoma; font-size: 16px; color: #000000;}
</style>

<script>
window.onload=function()
{
document.getElementById("bo").onclick=ins;
document.getElementById("im").onclick=pas;
};

function ins()
{
var v=document.getElementById("tx").value;
var sest=document.getElementById("tx").selectionStart;
var seen=document.getElementById("tx").selectionEnd;
var t=v.substring(sest, seen);
```

```
document.getElementById("tx").value=v.substring(0,
sest)+'<b>'+t+'</b>'+v.substring(seen);
}

function pas()
{
var v=document.getElementById("tx").value;
var sest=document.getElementById("tx").selectionStart;
var seen=document.getElementById("tx").selectionEnd;
document.getElementById("tx").value=v.substring(0, sest)+'<img
src="pict/comp.jpg" alt="Фото">'+v.substring(seen);
}
</script>
</head>

<body>

<input id="bo" type="button" value="BOLD">
<input id="im" type="button" value="IMG">
<br><br>
<textarea id="tx"></textarea>

</body>
</html>
```

WYSIWYG-редактор

Начать новую тему хотелось бы с обсуждения предыдущего примера. Главный недостаток нашего первого редактора в том, что все содержимое информационного блока предстает перед нами в виде HTML-кода. Поэтому, выполняя редактирование, невозможно представить, насколько удачно будут выглядеть его результаты. Только после того, как вы запишете изменения и просмотрите обновленную страницу, вы сможете делать выводы о качестве своей работы. А если оно вас не удовлетворит? Придется вновь редактировать информационный раздел. Так может повторяться по несколько раз – до тех пор, пока вы не получите желаемый результат.

Любой разработчик скажет, что это очень неудобно! Поэтому креативные программисты придумали не простой редактор, а WYSIWYG (или, как еще говорят, визуальный редактор). Он отображает редактируемые материалы практически в том же самом виде, как и на странице сайта. То есть, нажали кнопку «BOLD» – фрагмент текста действительно выделился жирным. Нажали кнопку «IMG» – и появилась настоящая фотография, а не ее HTML-код.

Термин WYSIWYG – это аббревиатура, произошедшая от английского выражения What You See Is What You Get, что в переводе означает «что вы видите, то вы и получаете». В нашем случае смысл этого выражения таков: что вы видите в редакторе, то и будет на странице сайта.

Хотя WYSIWYG – более «продвинутый» редактор, написать его не составит особого труда.

Итак, начнем. Первое, что мы сделаем – откажемся от поля textarea. Оно для WYSIWYG не подойдет – ведь любой код в textarea будет выглядеть как простой текст. Нам надо найти другое окно редактирования, которое, подобно браузеру, превращало бы каждый HTML-элемент в его визуальный аналог. И такое окно есть – это всем хорошо знакомый **iframe**.

Стоп, скажете вы. Любой HTML-код, загруженный во фрейм, действительно преобразуется в визуальное представление. Но как редактировать содержимое фрейма? Разве это возможно? Ответ – да. Чтобы превратить обычный iframe в окно редактирования, необходимо «включить» его атрибут **designMode**.

***Атрибут designMode**, когда он включен (**designMode="on"**), позволяет редактировать любую HTML-страницу. Действие designMode распространяется на весь документ. Эта особенность натолкнула программистов на идею создания визуальных редакторов. Если поместить документ в iframe, то можно менять прямо в браузере любую часть содержимого страницы. А если снабдить редактор кнопками для автоматического добавления тегов, то процесс редактирования становится очень похожим на то, что вы делаете в «Word». Благодаря визуальным редакторам сайтами могут управлять не только специалисты, но и обычные пользователи. В сети рекламируется множество платных и бесплатных CMS, с помощью которых сделать сайт может даже тот, кто совершенно не знаком с программированием. Непременный атрибут таких систем – визуальный редактор.*

Раз решение найдено, можно приступать к созданию редактора. Разместим на странице фрейм:

```
<iframe id="fr" name="fr"></iframe>
```

Атрибут src пока не указываем. Будем считать, что в **iframe** загружена новая, только что созданная страница, на которой пока ничего нет. Если вы примените этот редактор в своих разработках, в теге фрейма обязательно нужно будет указать адрес загружаемой страницы. Если вы хотите наполнить содержанием новую страницу, сначала создайте пустой файл, а уже потом открывайте его в редакторе. Не стоит действовать в обратном порядке: сначала в редакторе написать страницу, а потом сохранять ее как новый файл. В некоторых браузерах такой подход сработает, но Microsoft Edge может некорректно отображать фотографии во фрейме, у которого не указан src.

Над фреймом расположим кнопки автоматического добавления тегов. Так как новый редактор более серьезный, чем предыдущий, кнопок будет больше:

```
<input id="bo" type="button" value="BOLD">
<input id="it" type="button" value="ITALIC">
<input id="un" type="button" value="UNDERLINE">
<input id="im" type="button" value="IMG">
```

С их помощью мы можем делать текст не только жирным, но и наклонным, а также подчеркнутым.

Под фреймом у нас вновь появится поле **textarea**:

```
<textarea id="tx"></textarea>
```

Но теперь у него иное назначение, нежели раньше. Сейчас textarea выполняет двоякую функцию:

– во-первых, в поле мы можем видеть, как меняется HTML-разметка при добавлении в нее новых элементов;

– во вторых, поле выполняет роль контейнера, куда будет помещаться содержимое фрейма после завершения редактирования (наш визуальный редактор не снабжен функцией записи в файл или базу данных, чтобы исключить нежелательные действия недобросовестных посетителей).

Последний элемент страницы – символическая кнопка «ЗАПИСАТЬ», имитирующая сохранение результатов редактирования:

```
<input id="re" type="button" value="ЗАПИСАТЬ">
```

В реальности после ее нажатия финальное содержимое фрейма будет отображено в виде HTML-кода в поле textarea. Если вы решите воспользоваться примером для создания своего редактора, кнопке «ЗАПИСАТЬ» необходимо будет добавить действие, отправляющее данные из фрейма на сервер специальной программе.

Полностью страница редактора выглядит так:

```
<input id="bo" type="button" value="BOLD">
<input id="it" type="button" value="ITALIC">
<input id="un" type="button" value="UNDERLINE">
<input id="im" type="button" value="IMG">
<input id="re" type="button" value="ЗАПИСАТЬ">
<br><br><iframe id="fr" name="fr"></iframe>
<br><textarea id="tx"></textarea>
```

Посмотреть ее и поэкспериментировать с редактором можно здесь: http://bookjs.ru/w2.html. Также внешний вид страницы представлен на рисунке 4_3.

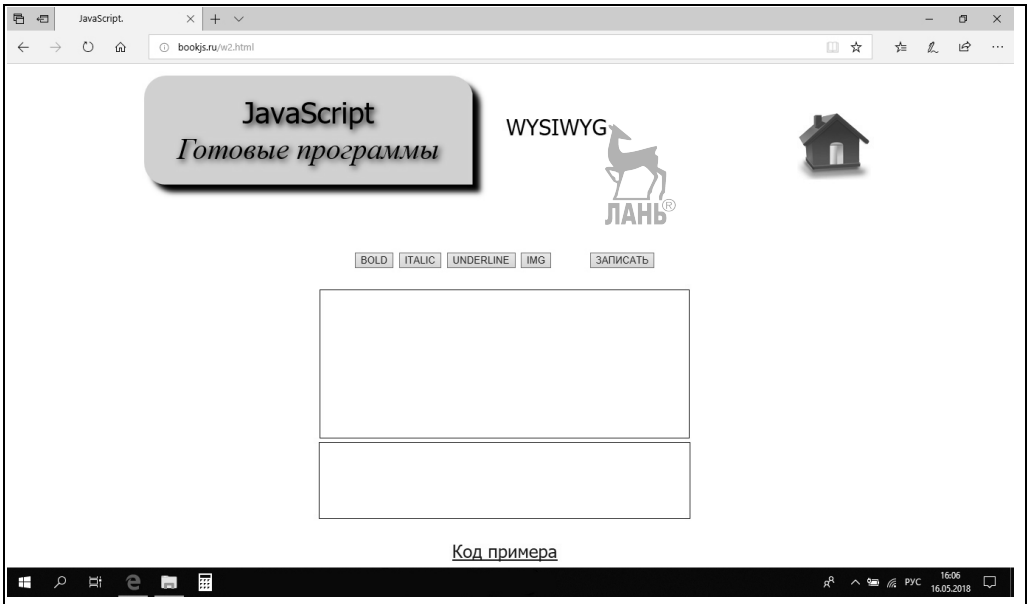

Рис. 4_3. WYSIWYG-редактор

Необходимые настройки окон записаны в таблице стилей:

```
iframe {width: 500px; height: 200px; border: 1px solid #0000CC;}
textarea {width: 500px; height: 100px; border: 1px solid #0000CC;
font-family: Tahoma; font-size: 14px; color: #000000;}
```

Свойства шрифта для **iframe** мы не указываем, так как по идее в него загружается страница со своими собственными настройками.

Попробуйте написать в верхнем окне какой-нибудь текст, отформатировать его, вставить изображение. Результаты ваших действий сразу отобразятся в textarea. Обратите внимание: в разных браузерах код, появляющийся в текстовом поле, может немного отличаться. Например, нажатие клавиши «Enter» приведет к появлению разных комбинаций тегов
 и   в разных браузерах (во всяком случае так было на момент написания книги).

Разберемся, как реализованы функциональные возможности нашего редактора.

Первым делом нужно «преобразовать» фрейм. Для этого сразу после загрузки страницы необходимо запустить функцию, включающую **designMode**:

```
function start()
{
frames["fr"].document.designMode="on";
}
```

Можно было бы инициировать запуск функции так:

```
window.onload= start;
```

Но у такого метода есть один недостаток: он позволяет выполнить только одну инструкцию. До сих пор это нам не мешало. Во всех предыдущих примерах после загрузки страницы мы либо сразу запускали единственную функцию (как в примерах со слайд-шоу), либо регистрировали обработчики событий (как в остальных примерах). А теперь необходимо одновременно запустить функцию **start()** и зарегистрировать обработчики событий. Поэтому мы используем новый метод – **addEventListener()**:

```
window.addEventListener("load", start);
```

Метод **addEventListener()** позволяет регистрировать любое количество обработчиков для одного события – в нашем случае для **load**.

Метод addEventListener *предназначен для регистрации обработчиков событий. Он принимает три аргумента, два из которых – обязательные: первый – событие (например, click или mouseover), второй – имя функции, обрабатывающей событие. Третий, необязательный аргумент, принимает логическое значение false или true и определяет, в какой фазе распространения события вызывается функция. Если третий аргумент не указан, он автоматически принимает значение false (то есть, обработчик регистрируется для фазы всплытия).*

Следующий этап – регистрация обработчиков для кнопок редактора и клавиатуры компьютера:

```
window.addEventListener("load", function()
{
frames["fr"].document.onkeyup=kep;
document.getElementById("bo").onclick=ins;
document.getElementById("it").onclick=emb;
document.getElementById("un").onclick=sti;
document.getElementById("im").onclick=pas;
document.getElementById("re").onclick=tra;
});
```

Теперь можно приступать к последовательному разбору функций.

Начнем с обработки нажатия клавиши. Для этих целей у нас есть функция **kep()**:

```
function kep()
{
document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
}
```

Установим курсор в редакторском окне и нажмем на клавиатуре любую букву. Она появится во фрейме и следом – в текстовом поле. Так происходит благодаря единственной инструкции, записанной в **kep()**. При нажатии любой клавиши эта инструкция присваивает содержимое фрейма текстовому полю в виде HTML-кода. До тех пор, пока мы вводим только буквы, в **textarea** никаких тегов нет. Теперь нажмем пробел или «Enter». В **textarea** появятся символ   или теги (какие – зависит от браузера). Каждое нажатие клавиши – это новая буква, символьный примитив или тег, появляющиеся в текстовом поле. Название функции **kep** произошло от события keypress. Тут нужно оговориться, что на самом деле после загрузки страницы у нас регистрируется обработчик события **keyup**, то есть функция выполняется после отпускания клавиши. Так сделано из-за особенностей «восприятия» браузерами события keypress – оно обрабатывается не в момент происхождения, а с отставанием на один шаг.

С клавиатурой все понятно. Переедем к кнопкам форматирования текста. Функции, «прикрепленные» к ним, работают по одному и тому же принципу, поэтому рассмотрим подробно только одну, например **ins()**. Как и в предыдущем редакторе, эта функция отвечает за выделение жирным фрагмента текста:

```
function ins()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Bold");
  document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
  }
}
```

Поскольку наш редактор гораздо профессиональнее, чем в предыдущем разделе, мы подойдем к форматированию более обстоятельно и первым делом проверим, выделен ли текст:

```
var docu=frames["fr"].document.getSelection();
```

Метод getSelection *возвращает объект Selection, который содержит выделенный пользователем текст. Разработчик имеет возможность на программном уровне манипулировать выделенным фрагментом: форматировать его, копировать, удалять, изменять.*

Если текст не выделен, а кнопка «BOLD» при этом была нажата, выведем стандартное окно диалога с предупреждением об ошибке:

```
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
```

Если выделение сделано, функция выполнит форматирование:

```
frames["fr"].document.execCommand("Bold");
```

Метод **execCommand** предоставляет набор различных команд для редактирования контента на странице, у которой включен режим **designMode**. В данном случае с помощью этого метода мы делаем жирным выделенный фрагмент текста.

Метод execCommand *получает в качестве аргумента имя команды. При простом форматировании обычно достаточно одного аргумента, например execCommand(«Bold»). Однако в некоторых случаях требуется передать три аргумента: имя команды; значение, указывающее, отображать или нет пользовательский интерфейс; адрес или текст. Второй аргумент обычно указывают false. Это означает, что третий аргумент передается в метод непосредственно из программы. Если указать второй аргумент true, то метод будет ожидать ввода третьего аргумента пользователем. Более аргументоемкие варианты* ***execCommand*** *применяются, например, для вставки ссылки или изображения в редактируемую страницу. В этих случаях сначала указывают имя, затем false, затем адрес ссылки или картинки. Метод* ***execCommand*** *создан давно. В связи с появлением HTML5 некоторые из ранее существовавших команд на сегодняшний день потеряли свою актуальность (например, fontSize).*

Последняя инструкция

```
document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
```

(она уже встречалась нам выше) присваивает содержимое фрейма текстовому полю в виде HTML-кода. В результате фрагмент текста обрамляется тегами .

Аналогично устроены функции, делающие текст наклонным и подчеркнутым:

```
function emb()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
```

```
  frames["fr"].document.execCommand("Italic");
  document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
  }
}

function sti()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Underline");
  document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
  }
}
```

Их названия образованы от английских слов stick и embed – вставить, встроить.

Функция **pas()** (от paste) добавляет в текст уже знакомое нам фото компьютера:

```
function pas()
{
var pict='<img src="pict/comp.jpg" alt="Фото">';
frames["fr"].document.execCommand("insertHTML", false, pict);
document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
}
```

Если вы установили курсор в окне редактирования, снимок появится в указанной точке (рис. 4_4), если нет – в большинстве браузеров он окажется в конце текста.

И снова уже знакомая нам инструкция:

```
function tra()
{
document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
}
```

Функция **tra()** (от transfer – передача) срабатывает при нажатии кнопки «ЗАПИСАТЬ» и передает HTML-код страницы из фрейма в текстовое поле.

Может возникнуть вопрос: зачем понадобились две совершенно идентичные функции: **kep()** и **tra()**? Нельзя ли обойтись одной? Дело в том, что в нашем примере кнопка «ЗАПИСАТЬ» и функция **tra()** – декоративные элементы,

только имитация настоящих. В реальном WYSIWYG-редакторе кнопка «ЗАПИСАТЬ» станет отправлять данные на сервер и в этом случае содержимое функции **tra()** будет кардинально отличаться от **kep()**. Поэтому мы оставим обе – на будущее.

Рис. 4_4. Редактор в процессе добавления текста и фотографии

Собрав отдельные части в единое целое, получим простой WYSIWYG-редактор:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>WYSIWYG</title>
<style>
iframe {width: 500px; height: 200px; border: 1px solid #0000CC;}
textarea {width: 500px; height: 100px; border: 1px solid #0000CC;
font-family: Tahoma; font-size: 14px; color: #000000;}
</style>

<script>
window.addEventListener("load", start);
window.addEventListener("load", function()
{
frames["fr"].document.onkeyup=kep;
```

```
document.getElementById("bo").onclick=ins;
document.getElementById("it").onclick=emb;
document.getElementById("un").onclick=sti;
document.getElementById("im").onclick=pas;
document.getElementById("re").onclick=tra;
});

function kep()
{
document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
}

function ins()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Bold");
  document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
  }
}

function emb()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Italic");
  document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
  }
}

function sti()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
```

```
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Underline");
  docu-
ment.getElementById("tx").value=frames["fr"].document.body.innerHTML;
  }
}

function pas()
{
var pict='<img src="pict/comp.jpg" alt="Фото">';
frames["fr"].document.execCommand("insertHTML", false, pict);
document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
}

function tra()
{
document.getElementById("tx").value=frames["fr"].document.body.innerHTML;
}

function start()
{
frames["fr"].document.designMode="on";
}
</script>
</head>

<body>

<input id="bo" type="button" value="BOLD">
<input id="it" type="button" value="ITALIC">
<input id="un" type="button" value="UNDERLINE">
<input id="im" type="button" value="IMG">
<input id="re" type="button" value="ЗАПИСАТЬ">
<br><br><iframe id="fr" name="fr"></iframe>
<br><textarea id="tx"></textarea>

</body>
</html>
```

Как уже было отмечено, метод **execCommand** позволяет редактировать страницу самыми разными способами: форматировать текст, добавлять графику, вставлять ссылки. Поэтому у желающих превратить наш экспериментальный

редактор в профессиональный инструмент есть все необходимое для совершенствования примера.

До и после

Помните, с чего мы начинали рассказ о WYSIWYG-редакторе? Одна из первых фраз была «Он отображает редактируемые материалы практически в том же самом виде, как и на странице сайта». В этом предложении оговорка «практически» сродни выражению «почти, да не совсем». Дело вот в чем. Окно фрейма имеет определенные размеры, которые могут совершенно не совпадать с размерами редактируемого блока на странице. Ширина окна в предыдущем примере – 500 пикселей. Теперь представьте себе такую ситуацию: вы работаете с информационным блоком из первого примера этой главы. Редактируя документ, вы добавили в него текст и фотографии, гармонично скомпоновали все элементы в окне шириной 500 пикселей, записали результат в файл. А затем посмотрели готовую страницу и обнаружили, что этот результат выглядит совсем не так, как в окне фрейма, потому что ширина информационного блока на странице – 660 пикселей. В итоге элементы блока растянулись и заняли совершенно иное положение. Приходится вновь загружать страницу в редактор и менять компоновку до тех пор, пока не получится желаемый результат.

Когда-то давно, создавая первые WYSIWYG-редакторы, я тоже столкнулся с подобной проблемой. И тогда придумал редактор, который бы позволял видеть на странице результат работы еще до его записи в файл.

Чтобы понять, как устроен такой редактор, зайдите на страницу http://bookjs.ru/w3.html демонстрационного сайта или посмотрите на рисунок 4_5.

Представим, что у нас есть ресурс, посвященный астрономии. Страница такого сайта имеет шапку, меню, рекламный блок справа и информационный блок слева. Наша задача – отредактировать статью, посвященную планете Марс.

Как сделать, чтобы наши правки сразу выглядели «натурально»? Во-первых, надо поместить под редактором копию данной страницы. Во-вторых, в информационный блок этой копии передавать код из фрейма при каждом нашем действии. Вставили фотографию – и сразу видно, как она будет смотреться на странице. Отформатировали текст – и сразу видно, хорошо ли у нас получилось. Добились нужного результата – можно записывать его в файл.

Добавить копию документа на страницу редактора можно двумя способами. Если у вас все разделы сайта оформлены по единому шаблону, надо под редактором поместить контейнер div, в него добавить шаблон и еще один div, который расположить там, где находится информационный блок. В этот второй div будет помещаться код из редактора. Так сделано в нашем примере, который мы чуть позже разберем подробно. Ели вы не хотите привязывать свой редактор к одному единственному шаблону, можно прибегнуть к помощи серверной программы. Она будет брать HTML-файл, извлекать код из тега body, добавлять в нужном месте контейнер div и помещать копию страницы под окно фрейма.

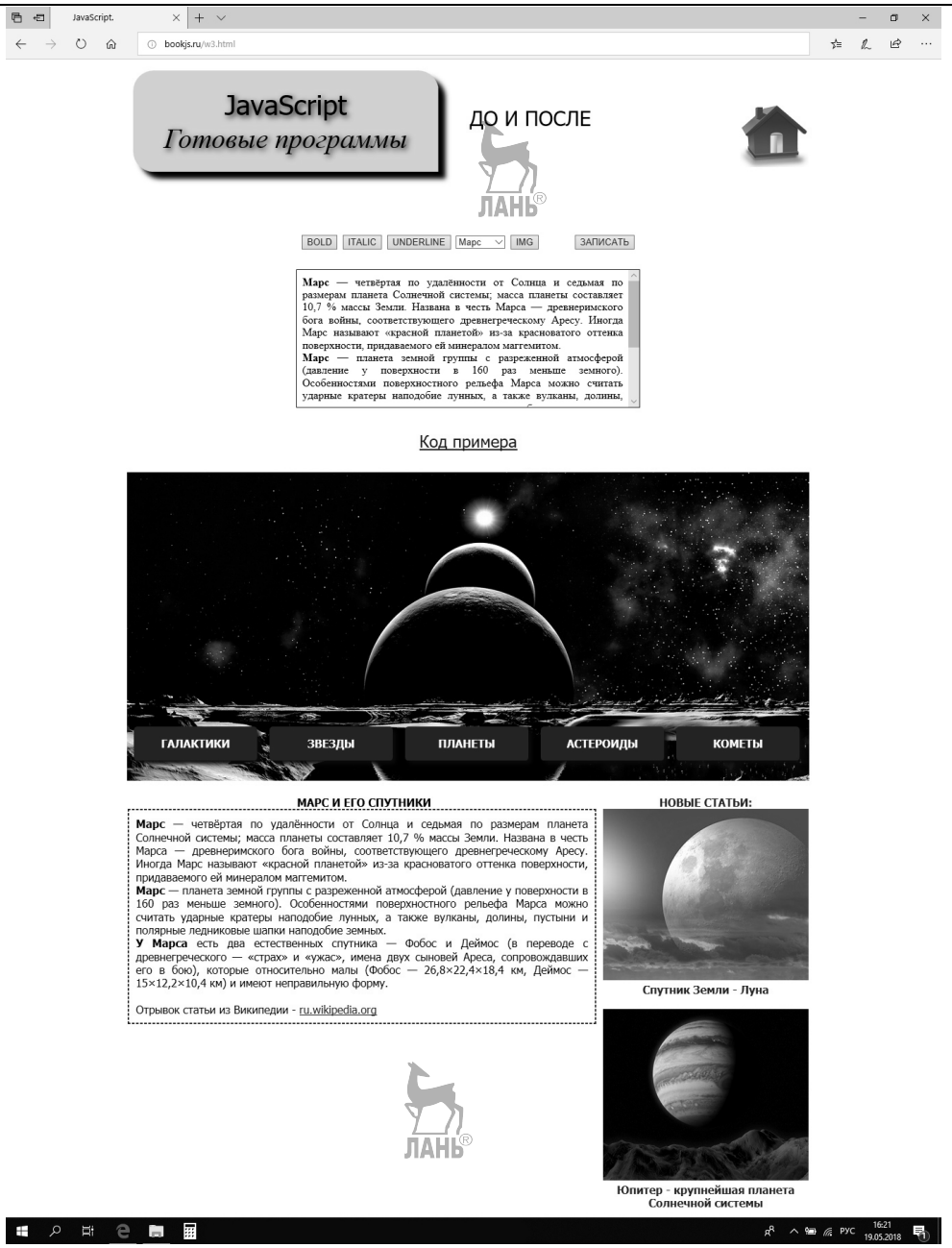

Рис. 4_5. WYSIWYG-редактор с функцией предпросмотра

Вообще, без помощи серверных программ WYSIWYG-редактор обойтись не может. Эти программы нужны не только, чтобы записать изменения, внесен-

ные вами в документ. Если вы приступаете к работе с новой страницей, ее информационный блок пуст и загружать в редактор ничего не надо. Если же у страницы информационный блок заполнен и вам требуется внести изменения, содержание блока необходимо загрузить в iframe. Для этого тоже нужна серверная программа.

Перейдем к устройству редактора с функцией предпросмотра. Теперь, когда принцип его действия понятен, упростим наш пример. Для начала избавимся от текстового поля – механизм передачи данных в textarea мы уже разобрали, повторяться не имеет смысла. Затем откажемся от шаблона – будем считать, что каким-то из описанных способов мы поместили его на странице. Заодно представим, что блок информации тоже загружен в шаблон. В разбираемом примере оставим только контейнер блока информации, куда будем помещать код из редактора:

```
<div id="cont"></div>

#cont {width: 660px;}
```

Добавим немного функциональности, предоставив разработчику возможность выбирать фотографии из нескольких имеющихся на сервере:

```
<select id="sel"><option selected value="cosmos/ma.jpg">Марс</option>
<option value="cosmos/fo.jpg">Фобос</option>
<option value="cosmos/de.jpg">Деймос</option></select>
<input id="im" type="button" value="IMG">
```

Остальные элементы останутся теми же, что и в первой версии WYSIWYG-редактора:

```
<input id="bo" type="button" value="BOLD">
<input id="it" type="button" value="ITALIC">
<input id="un" type="button" value="UNDERLINE">

<input id="re" type="button" value="ЗАПИСАТЬ">

<iframe id="fr" name="fr"></iframe>

iframe {width: 500px; height: 200px; border: 1px solid #0000CC;}
```

Как и в предыдущей версии редактора, регистрируем те же самые обработчики событий:

```
<script>
window.addEventListener("load", start);
window.addEventListener("load", function()
```

```
{
frames["fr"].document.onkeyup=kep;
document.getElementById("bo").onclick=ins;
document.getElementById("it").onclick=emb;
document.getElementById("un").onclick=sti;
document.getElementById("im").onclick=pas;
document.getElementById("re").onclick=tra;
});
```

В функциях произошли некоторые изменения. Во-первых, в функцию **start()** добавлена инструкция по передаче данных из контейнера во фрейм. Мы с вами договорились считать, что информационный блок загружен в шаблон. Значит, надо получить его содержимое и передать в окно редактирования (при этом неважно – блок пустой или уже содержит какую-то информацию):

```
frames["fr"].document.body.innerHTML=document.getElementById("cont").
innerHTML;
```

Во-вторых, во всех функциях появилась новая инструкция для передачи HTML-кода из фрейма. Раньше данные отправлялись в textarea, теперь в контейнер **div**:

```
document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
```

Заметно изменилась функция **pas()**:

```
function pas()
{
var pict='<img src="'+document.getElementById("sel").value+'" alt="Фото">';
frames["fr"].document.execCommand("insertHTML", false, pict);
document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
}
```

Появилась новая версия первой строки:

```
var pict='<img src="'+document.getElementById("sel").value+'" alt="Фото">';
```

Здесь мы определяем, какое фото выбрано из выпадающего списка и присваиваем переменной **pict** адрес данного снимка.

Полный код WYSIWYG-редактора с предпросмотром выглядит так:

```
<!DOCTYPE html>
<html lang="ru">
<head>
```

```
<meta charset="utf-8">
<title>До и после</title>
<style>
iframe {width: 500px; height: 200px; border: 1px solid #0000CC;}
#cont {width: 660px;}
</style>

<script>
window.addEventListener("load", start);
window.addEventListener("load", function()
{
frames["fr"].document.onkeyup=kep;
document.getElementById("bo").onclick=ins;
document.getElementById("it").onclick=emb;
document.getElementById("un").onclick=sti;
document.getElementById("im").onclick=pas;
document.getElementById("re").onclick=tra;
});

function kep()
{
document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
}

function ins()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Bold");
  document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
  }
}

function emb()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
```

```
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Italic");
  document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
  }
}

function sti()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("Underline");
  document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
  }
}

function pas()
{
var pict='<img src="'+document.getElementById("sel").value+'" alt="Фото">';
frames["fr"].document.execCommand("insertHTML", false, pict);
document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
}

function tra()
{
document.getElementById("cont").innerHTML=frames["fr"].document.body.
innerHTML;
}

function start()
{
frames["fr"].document.designMode="on";
frames["fr"].document.body.innerHTML=document.getElementById("cont").inn
erHTML;
}
```

```
</script>
</head>

<body>

<input id="bo" type="button" value="BOLD">
<input id="it" type="button" value="ITALIC">
<input id="un" type="button" value="UNDERLINE">
<select id="sel"><option selected value="cosmos/ma.jpg">Марс</option>
<option value="cosmos/fo.jpg">Фобос</option>
<option value="cosmos/de.jpg">Деймос</option></select>
<input id="im" type="button" value="IMG">
<input id="re" type="button" value="ЗАПИСАТЬ">
<br><br><iframe id="fr" name="fr"></iframe>

<div id="cont"></div>

</body>
</html>
```

Прочитав эту главу, вы могли убедиться, что создание редакторов контента – не самое трудное занятие. На мой взгляд, сложнее «скооперировать» такие редакторы с серверными программами, которые будут управлять загрузкой во фреймы страниц или их отдельных частей, выполнять встраивание копий шаблонов, записывать конечные результаты в файлы или базы данных. Можно было бы привести примеры таких программ, но, как и в некоторых предыдущих случаях, констатирую, что это не входит в задачи книги. Поэтому просто еще раз повторю совет, который уже давал читателю: изучите один из языков web-программирования – и вы сможете писать необходимые приложения самостоятельно.

Редактор PRO

Термин «PRO» наверняка не раз встречался читателю. Он произошел от сокращения английского слова professional. Маркером «PRO» обычно помечают программы, устройства и технологии, которые рассчитаны на получение результатов высокого уровня. Проще говоря, так обозначают высокопрофессиональные, квалифицированные разработки.

В этом разделе у нас пойдет речь как раз о такой программе – WYSIWYG-редакторе уровня PRO (или, если быть кристально объективными, почти PRO – а почему «почти», объясним в самом конце этой темы).

Главная особенность нового ПО – наличие широких возможностей для создания и редактирования контента. В последней версии редактора у нас было четыре кнопки: три для форматирования текста и одна для добавления фото-

графий. Теперь кнопок будет шестнадцать. А это значит, что функциональные возможности редактора выросли в четыре раза! Тем самым мы почти полностью перекрываем все мыслимые случаи форматирования текста, а кроме того добавляем возможность размещать на странице помимо фото еще ссылки и таблицы.

Версию такого редактора читатели могут увидеть на странице http://bookjs.ru/w4.html или на рисунке 4_6. С первого взгляда ясно, что перед вами усовершенствованный WYSIWYG с предпросмотром страниц.

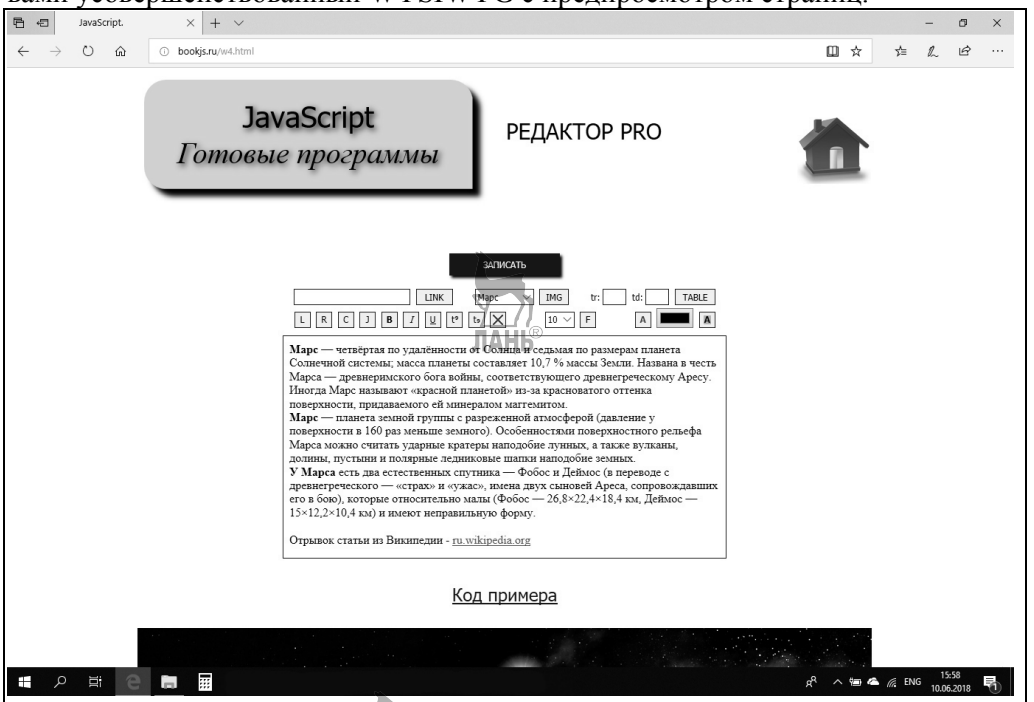

Рис. 4_6. Редактор PRO – профессиональный или продвинутый – как кому больше нравится

Разберемся, какие новые функции у нас есть теперь:

– размещение таблиц и ссылок (http, https или mailto) – про это мы уже сказали;

– выравнивание текста по левому или правому краю, по центру и по ширине;

– оформление символов или слов в виде надстрочных или подстрочных;

– установка размера шрифта (в пикселях);

– изменение цвета шрифта;

– изменение цвета фона под текстом или отдельными словами;

– удаление форматирования из выделенного фрагмента или целого текста.

Забегая вперед, скажу: несмотря на обилие функциональных возможностей, это не самая сложная программа в данной книге. Но зато самая длинная. Поэтому я решил поступить следующим образом:

– не давать полный листинг редактора – желающие могут посмотреть и скопировать его, перейдя по ссылке «Код примера»;

– полностью описать разметку страницы, чтобы было понятно, какие кнопки отвечают за те или иные возможности;

– не давать описание всех функций, а разобрать только 8 основных – остальные действуют похожим образом.

Как всегда, начинаем с того, что добавляем на страницу необходимые элементы. В первую очередь, у нас будет уже знакомый по предыдущим версиям фрейм:

<iframe id="fr" name="fr"></iframe>

Только теперь мы увеличим его размеры:

iframe {width: 600px; height: 300px; border: 1px solid #0000CC;}

Затем у нас есть контейнер блока информации

** **
и кнопка записи отредактированного содержимого

<input id="re" type="button" value="ЗАПИСАТЬ">

Кнопочная «палитра» теперь гораздо многообразнее:

```
<input id="tex" type="text" maxlength="100">
<input id="lin" type="button" value="LINK" title="Вставить ссылку">
<select id="sel"><option selected value="cosmos/ma.jpg">Марс</option>
<option value="cosmos/fo.jpg">Фобос</option>
<option value="cosmos/de.jpg">Деймос</option></select>
<input id="im" type="button" value="IMG" title="Вставить изображение">
tr: <input id="trs" type="text" maxlength="1">
td: <input id="tdk" type="text" maxlength="1">
<input id="tabl" type="button" value="TABLE" title="Вставить таблицу"><br>
<input id="le" type="button" value="L" title="Выровнять по левому краю">
<input id="ri" type="button" value="R" title="Выровнять по правому краю">
<input id="ce" type="button" value="C" title="Выровнять по середине">
<input id="ju" type="button" value="J" title="Выровнять по ширине">
<input id="bo" type="button" value="B" title="Выделить жирным">
<input id="it" type="button" value="I" title="Выделить курсивом">
<input id="un" type="button" value="U" title="Подчеркнуть">
```

```
<input id="tv" type="button" value="t&#8313;" title="Сделать надстрочным">
<input id="tn" type="button" value="t&#8329;" title="Сделать подстрочным">
<input id="remo" type="button" value=" " title="Удалить форматирование">
<select id="sfo"><option selected value="10">10</option>
<option value="11">11</option>
<option value="12">12</option>
<option value="13">13</option>
<option value="14">14</option>
<option value="15">15</option>
<option value="16">16</option>
<option value="17">17</option>
<option value="18">18</option>
<option value="19">19</option>
<option value="20">20</option>
<option value="21">21</option>
<option value="22">22</option>
<option value="23">23</option>
<option value="24">24</option>
<option value="20">25</option>
<option value="21">26</option>
<option value="22">27</option>
<option value="23">28</option>
<option value="24">29</option>
<option value="24">30</option></select>
<input id="fs" type="button" value="F" title="Изменить размер шрифта">
<input id="co" type="button" value="A" title="Изменить цвет шрифта">
<input id="ccc" type="color" title="Нажмите, чтобы выбрать цвет">
<input id="colo" type="button" value="A" title="Изменить фон шрифта">
```

Назначение каждого элемента **button** поясняет всплывающая подсказка, которая появляется при наведении указателя мыши на кнопку.

Оформление различных полей редактора выглядит следующим образом:

```
#bo {font-weight: bold;}
#it {font-style: italic;}
#un {text-decoration: underline;}
#co {color: #CC0000; font-weight: bold;}
#colo {font-weight: bold; background: url(img/colo.jpg);}
#remo {background: url(img/remo.jpg);}
#trs {width: 30px;}
#tdk {width: 30px;}
```

Так как для двух кнопок необходимые пиктограммы проблематично создать средствами HTML, мы добавили им фоновые рисунки:

```
#colo {…; background: url(img/colo.jpg);}
#remo {background: url(img/remo.jpg);}
```

Все необходимые компоненты размещены на странице, настало время зарегистрировать обработчики событий:

```
window.addEventListener("load", start);
window.addEventListener("load", function()
{
frames["fr"].document.onkeyup=kep;
document.getElementById("bo").onclick=ins;
document.getElementById("it").onclick=emb;
document.getElementById("un").onclick=sti;
document.getElementById("im").onclick=pas;
document.getElementById("re").onclick=tra;
document.getElementById("fs").onclick=fos;
document.getElementById("tv").onclick=ftv;
document.getElementById("tn").onclick=ftn;
document.getElementById("co").onclick=clr;
document.getElementById("colo").onclick=bgk;
document.getElementById("le").onclick=lef;
document.getElementById("ri").onclick=rig;
document.getElementById("ce").onclick=cen;
document.getElementById("ju").onclick=jus;
document.getElementById("remo").onclick=rem;
document.getElementById("lin").onclick=lin;
document.getElementById("tabl").onclick=tab;
});
```

Так как самый первый элемент нашего редактора – кнопка добавления ссылки, начнем с функции **lin()**:

```
function lin()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  if(document.getElementById("tex").value.length<10)
```

```
    {
    alert("Вы не ввели ссылку!");
    }
  else
    {
    var li='<a href="'+document.getElementById("tex").value+'" target="_blank">'+docu+'</a>';
    frames["fr"].document.execCommand("insertHTML", false, li);
    document.getElementById("cont").innerHTML=frames["fr"].document.body.innerHTML;
    }
  }
}
```

Администратору необходимо ввести в поле **tex** адрес ссылки (включая наименование протокола – http, https или mailto), выделить одно или несколько слов в тексте и нажать кнопку LINK.

Первым делом функция проверит, выделен ли текст:

```
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
```

Затем выполняется проверка, введен ли адрес ссылки:

```
if(document.getElementById("tex").value.length<10)
  {
  alert("Вы не ввели ссылку!");
  }
```

Если все в порядке, ничего не пропущено, ссылка добавляется в текст:

```
var li='<a href="'+document.getElementById("tex").value+'" target="_blank">'+docu+'</a>';
frames["fr"].document.execCommand("insertHTML", false, li);
    document.getElementById("cont").innerHTML=frames["fr"].document.body.innerHTML;
```

Принцип вставки картинки нам известен из предыдущих примеров, поэтому пропустим функцию **pas()** и перейдем к созданию таблицы.

Для начала в поле **trs** необходимо указать количество строк будущей таблицы, а в поле **tdk** – количество столбцов. В нашем примере эти значения ограничены числом 9 (так как максимальная длина полей **trs** и **tdk** – один символ, то двухзначное число ввести не получится). Естественно, что вы, перенося ре-

дактор в свой проект, можете указать иной верхний предел для количества строк и колонок.

После нажатия кнопки TABLE запускается функция **tab()**:

```
function tab()
{
if((document.getElementById("trs").value.length<1)||(document.getElementById
d("tdk").value.length<1))
  {
  alert("Вы не указали количество строк или столбцов!");
  }
else
  {
  var tb='<table style="width: 650px;">';
  var trs=document.getElementById("trs").value;
  var tdk=document.getElementById("tdk").value;
  for(var i=0; i<trs; i++)
    {
    tb+='<tr>';
    for(var k=0; k<tdk; k++)
      {
      tb+='<td style="border: 1px solid #000000;"> </td>';
      }
    tb+='</tr>';
    }
  tb+='</table>';
  frames["fr"].document.execCommand("insertHTML", false, tb);
  document.getElementById("cont").innerHTML=frames["fr"].document.
body.innerHTML;
  }
}
```

Первым делом проверяется, введены ли значения для количества строк и столбцов. Если вы пропустили один из параметров, откроется диалоговое окно с предупреждением. Если данные введены правильно, запускаются два цикла **for** (один внутри другого), в которых происходит формирование тела таблицы. Сначала создается открывающий дескриптор **table**:

```
var tb='<table style="width: 650px;">';
```

Затем во внешнем цикле происходит формирование строк:

```
for(var i=0; i<trs; i++)
    {
    tb+='<tr>';
```

```
…
tb+='</tr>';
}
```

а во внутреннем – формирование колонок:

```
for(var k=0; k<tdk; k++)
  {
  tb+='<td style="border: 1px solid #000000;"> </td>';
  }
```

Завершается процесс добавлением закрывающего тега:

```
tb+='</table>';
```

Теперь готовая таблица вставляется в **iframe**:

```
frames["fr"].document.execCommand("insertHTML", false, tb);
```

Некоторые комментарии к функции **tab()**. Мы создаем таблицу определенного внешнего вида. Если он не устраивает вас, можно внести необходимые изменения в свойства стилей. Размер таблицы 650 пикселей выбран, исходя из необходимости целиком поместить ее в информационный блок страницы, рассказывающей о планете Марс. Нельзя сделать таблицу без рамок – иначе она потеряется во фрейме и вы не сможете найти ячейки для вставки в них данных. В описании функции опущены проверки ввода символов в полях **trs** и **tdk** (а в них можно вставить цифру 0 или любую букву, что приведет к возникновению ошибок). Сделано это по следующей причине: администратор редактирует страницу собственного сайта и вряд ли захочет навредить своему творению.

Действие кнопок выравнивания текста рассмотрим на примере функции lef() (выравнивание по левому краю). Она очень простая:

```
function lef()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  var di='<div style="text-align: left;">'+docu+'</div>';
  frames["fr"].document.execCommand("insertHTML", false, di);
  document.getElementById("cont").innerHTML=frames["fr"].document.
body.innerHTML;
```

```
  }
}
```

Выделили текст, нажали кнопку L и соответствующий блок текста выполнит «равнение налево». Как видно из описания функции, она добавляет к выделенному фрагменту тег div. Почему именно так? Мы уже выяснили во втором разделе этой главы, что именно такие теги формируются в большинстве браузеров при создании новой строки или абзаца. Так что будем следовать в фарватере этих браузеров.

Следующие на очереди функции – «превращающие» отдельные символы или целые слова в надстрочные или подстрочные. Их действие рассмотрим на примере функции **ftv()**. Она делает выделенный фрагмент надстрочным:

```
function ftv()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  var tv='<sup>'+docu+'</sup>';
  frames["fr"].document.execCommand("insertHTML", false, tv);
  document.getElementById("cont").innerHTML=frames["fr"].document.
body.innerHTML;
  }
}
```

Как видите, функция очень простая. Она добавляет к символу или слову теги ****.

Очистку форматирования выполняет функция **rem()**:

```
function rem()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  frames["fr"].document.execCommand("RemoveFormat");
  document.getElementById("cont").innerHTML=frames["fr"].document.
body.innerHTML;
  }
```

```
}
```

Она еще проще – просто выполняет команду **RemoveFormat** по отношению к выделенному блоку текста.

Теперь на очереди процесс изменения размера шрифта. Им управляет функция **fos()**:

```
function fos()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  var fons='<span style="font-size: '+document.getElementById("sfo").value+
'px;">'+docu+'</span>';
  frames["fr"].document.execCommand("insertHTML", false, fons);
  document.getElementById("cont").innerHTML=frames["fr"].document.
body.innerHTML;
  }
}
```

Функция заключает фрагмент текста в тег span, в стиле которого указан выбранный вами размер шрифта (напоминаю – в пикселях – для идентичного отображения в разных браузерах).

Последние две функции, из тех, что мы будем рассматривать, работают с цветом.

Сначала необходимо щелкнуть на элементе выбора цветовой гаммы. Откроется окно с палитрой красок (внешний вид такой палитры отличается в разных браузерах). Выполнив необходимые настройки, нажмите кнопку, подтверждающую выбор. Теперь выделите часть текста или слово и нажмите левую кнопку с красной буквой А. Текст поменяет цвет. Произошло это благодаря функции **clr()**:

```
function clr()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
```

```
  var cl='<span style="color: '+document.getElement-
ById("ccc").value+';">'+docu+'</span>';
  frames["fr"].document.execCommand("insertHTML", false, cl);
  docu-
ment.getElementById("cont").innerHTML=frames["fr"].document.body.inner
HTML;
  }
}
```

Как и в предыдущем случае, функция заключает фрагмент текста в тег span, в стиле которого указан выбранный вами цвет шрифта.

Если же вы нажмете кнопку с буквой А на оранжевой «подкладке», то изменится цвет фона под выделенный текстом. В этом случае добавляется тег span, в стиле которого выбранный вами цвет присваивается свойству **background-color**:

```
function bgk()
{
var docu=frames["fr"].document.getSelection();
if(docu==0)
  {
  alert("Вы не выделили текст!");
  }
else
  {
  var cl='<span style="background-color:
'+document.getElementById("ccc").value+';">'+docu+'</span>';
  frames["fr"].document.execCommand("insertHTML", false, cl);
  docu-
ment.getElementById("cont").innerHTML=frames["fr"].document.body.inner
HTML;
  }
}
```

Думаю, читатели уже убедились в справедливости моего тезиса о том, что наш продвинутый редактор – не самая сложная программа. Однако именно из-за этого она немного недотягивает до того, чтобы гордо именоваться истинно профессиональной.

Во-первых, не помешало бы увеличить перечень настроек, которые способен выполнять администратор, добавляя ссылки, картинки и таблицы. Например, фото можно вставлять на страницу не только с вашего сервера, но и со сторонних ресурсов. Можно добавлять к изображению рамки разного цвета и толщины, подписи (alt и title), позиционирование. У таблицы настраивать ее ширину, шрифт, рамку, фон, внутренние и внешние зазоры. Ссылку делать с подчеркиванием или без, открывать в том же окне или в новом. Кстати, к выбо-

ру размера шрифта желательно добавить выбор гарнитуры (лучше использовать наиболее распространенные, типа Arial, Tahoma и тому подобные, так как на компьютере посетителя скорее всего не окажется каких-то особо стильных шрифтов).

Во-вторых, можно увеличить список размещаемых элементов. На страницу легко добавить функции вставки маркированных списков, горизонтальных линий, специальных символов (например, ©, @), элементов форм. Не будет лишней кнопка удаления ссылки (без удаления текста, который был в нее «обернут»). Можно также добавить под фреймом поле textarea (как мы делали это во втором разделе данной главы), чтобы видеть код страницы (и при необходимости корректировать его).

Думаю, перечисление функционала, которого не хватает редактору, вполне может побудить читателей усовершенствовать пример и сделать еще более крутой WYSIWYG.

Глава 5.
Разные полезные рецепты

В этой главе собрано шесть разных программ. Все они отличаются уровнем сложности. Здесь вы найдете самый простой скрипт, состоящий всего из нескольких строк, и самые объемные – настолько, что я не стал приводить в книге их полный код. Желающие могут скопировать эти «навороченные» примеры на сайте поддержки http://bookjs.ru/.

В предыдущих главах наша основная задача была создать код, который можно легко внедрить в любой собственный проект. Теперь у нас иная цель: предложить не готовые программы, а методы подбора «ингредиентов» для ситуаций, в которых возможны самые разные решения.

Удаление файла

До сих пор во всех главах книги речь шла о том, как добавить на сайт какие-либо данные. Сначала мы вставляли фотографии, затем размещали меню, потом закачивали сообщения от посетителей. Наконец, в четвертой главе создавали контент на странице. Однако нередко администратору сайта необходимо, наоборот, что-то удалить. Например, файл, потерявший актуальность, или баннер, срок размещения которого закончился. Иногда и сами клиенты могут удалять лишнюю информацию, если речь идет о портале или социальной сети, где у них есть персональные страницы. В этом разделе мы коснемся как раз таких ситуаций.

Конечно, разбирать все возможные варианты – безнадежная затея. Поэтому, сузим проблему и поговорим в качестве примера о каком-нибудь частном случае. Предположим, о необходимости удалить с сервера лишние фото. На этом примере и продемонстрируем алгоритм решения задачи.

Тут возможны два разных подхода.

Первый – наиболее простой. Создаем выпадающий список, а рядом с ним кнопку «Удалить», нажатием на которую вызывается процесс «ликвидации» выбранного снимка. В таком варианте даже JavaScript не нужен – настолько все просто. Но, как известно, часто в простоте таится не только красота, но и подводные камни. Беда в том, что администратор может по ошибке выбрать не то изображение, а нажатие кнопки безвозвратно ликвидирует файл. Сайт – не компьютер, у него нет корзины для мусора, из которой можно восстановить ошибочно удаленную фотографию. Поэтому более разумно будет при нажатии на кнопку отложить процесс и задать администратору вопрос: вы точно хотите удалить этот снимок? Так появляется возможность исправить оплошность. Ответил нет – и выполнение программы будет прервано, ответил да – удаление состоится.

Для реализации такого похода понадобится JavaScript. Мы напишем небольшой скрипт, который станет запускать диалоговое окно с соответствующим вопросом и кнопками для положительного и отрицательного ответа.

JavaScript предлагает разработчику удобный метод confirm(), который выводит стандартное окно с сообщением и двумя кнопками: ОК и «Отмена». Можно было бы применить этот метод, но мы не станем прибегать к нему. Основная причина – внешний вид таких окон во всех браузерах довольно примитивен и в подавляющем большинстве случаев никак не вяжется с дизайном сайта. Профессиональные разработчики почти никогда не используют это окно. Чаще всего программисты создают свои диалоговые окна, внешний вид которых органично вписывается в оформление страницы. Пойдем этим путем и мы.

Обращаю внимание: диалоговое окно, использованное в нашем примере – это всего лишь один вариант из бесконечного множества других. Вы, копируя пример, можете изменить размеры и оформление окна (а также надписи кнопок) по своему вкусу.

Приступим.

Первым делом выведем список фотографий, загруженных на сервер. Для этой цели используем снимки, которые фигурировали в примере визуального редактора с режимом предпросмотра. Получится так:

```
<select id="sel"><option selected value="cosmos/ma.jpg">Марс</option>
<option value="cosmos/fo.jpg">Фобос</option>
<option value="cosmos/de.jpg">Деймос</option></select>
```

Добавим кнопку «Удалить»:

```
<input id="bu" type="button" value="Удалить">
```

Теперь создадим диалоговое окно:

```
<div id="lay"><p>Удалить изображение?<br><br>
<input id="but1" type="button" value="  НЕТ  ">    
<input id="but2" type="button" value="   ДА   "></p></div>
```

Оно размещается в отдельном слое. В его верхней части выводится сообщение «Удалить изображение?», а ниже располагаются две кнопки «НЕТ» и «ДА». Пробелы перед текстом и после текста кнопок нужны, чтобы растянуть кнопки по ширине, не «заморачиваясь» настройками стилей (тем самым мы максимально упрощаем пример). Пробелы между кнопками необходимы для создания зазора между ними (так тоже упрощается оформление). Используя пример в своих проектах, вы можете удалить пробелы и настроить все необходимые размеры с помощью таблицы стилей.

Параметры диалогового окна:

```
#lay {border: 1px solid #000000; visibility: hidden; position: absolute;
z-index: 55; background: #FFFFFF; padding: 10px;}
```

В исходном состоянии окно скрыто от пользователя.

Сообщение и кнопки выравниваем по центру:

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

Под списком и кнопкой расположим текст оповещения о состоянии процесса:

```
<span id="att"><br><br>Идёт удаление...</span>
```

Переводы строк перед сообщением мы добавили, чтобы отделить надпись от верхних элементов. В исходном состоянии оповещение скрыто:

```
#att {display: none; color: #CC0000;}
```

Прежде чем приступить к написанию кода, обсудим один важный момент. Дело в том, что в примере, представленном на странице http://bookjs.ru/v1.html, инструкции удаления файла отсутствуют. Это сделано по понятным причинам: чтобы не дать посетителям возможность ликвидировать изображения. В коде примера такая инструкция есть. В ней присутствует ссылка на абстрактный серверный скрипт (который на самом деле не существует). При необходимости читатели сами заменят имя и адрес скрипта на реальные.

Внешний вид нашего окна диалога показан на рисунке 5_1.

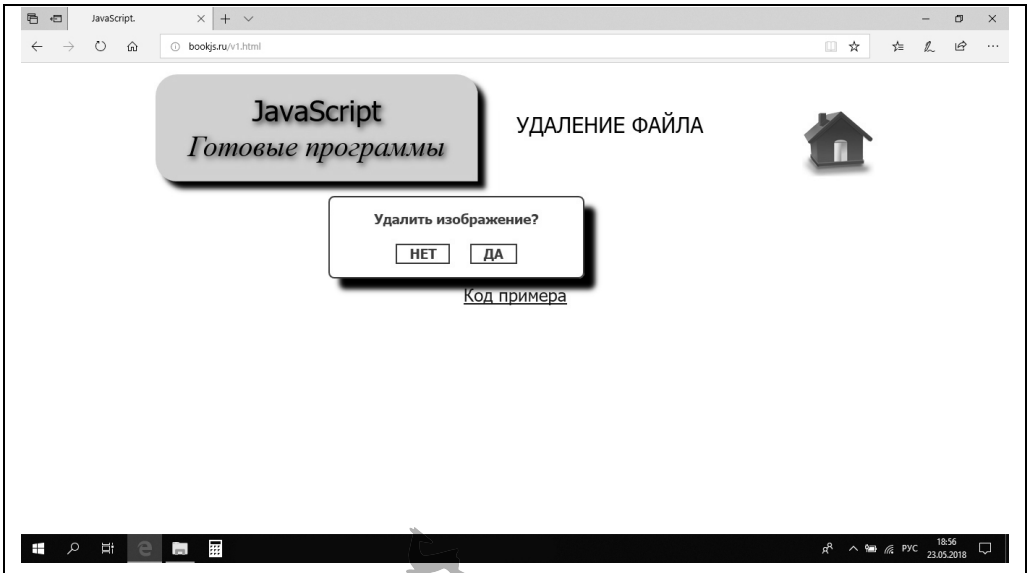

Рис. 5_1. «Самодельное» диалоговое окно

Для удаления файла нам понадобятся три функции:

```
window.onload=function()
{
document.getElementById("bu").onclick=rem;
document.getElementById("but1").onclick=reta;
document.getElementById("but2").onclick=era;
};
```

Первая – **rem()** (от remove – удалить) – нужна, чтобы определить точку клика и открыть в соответствующем месте диалоговое окно. Делается это методом, уже известным нам из главы, посвященной меню:

```
function rem(event)
{
var e=event;
var x=e.clientX+window.scrollX;
var y=e.clientY+window.scrollY;
document.getElementById("lay").style.left=x+"px";
document.getElementById("lay").style.top=y+"px";
document.getElementById("lay").style.visibility="visible";
}
```


Вторая функция – **reta()** (от retard – тормозить) – скрывает диалоговое окно, если нажата кнопка «НЕТ»:

```
function reta()
{
document.getElementById("lay").style.visibility="hidden";
}
```

Самые главные действия сосредоточены в функции **era()** (от erase – стирать). Она запускается, когда клиент подтвердил удаление картинки:

```
function era()
{
document.getElementById("lay").style.visibility="hidden";
document.getElementById("att").style.display="inline";
window.location="del.php?del="+document.getElementById("sel").value;
}
```

Первая инструкция

```
document.getElementById("lay").style.visibility="hidden";
```

скрывает диалоговое окно. Вторая

document.getElementById("att").style.display="inline";

выводит на страницу сообщение "Идёт удаление...". Третья

window.location="del.php?del="+document.getElementById("sel").value;

обращается к серверной программе **del.php**, которая удаляет файл. Имя и адрес файла передается в строке запроса:

"del.php?del="+document.getElementById("sel").value

Свойство location *содержит адрес текущей страницы, загруженной в браузер. Чтобы загрузить другую страницу, необходимо присвоить свойству новый адрес.*

У нас получилась довольно простая программа:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Удаление файла</title>
<style>
#lay {border: 1px solid #000000; visibility: hidden; position: absolute;
z-index: 55; background: #FFFFFF; padding: 10px;}
p {text-align: center;}
#att {display: none; color: #CC0000;}
</style>

<script>
window.onload=function()
{
document.getElementById("bu").onclick=rem;
document.getElementById("but1").onclick=reta;
document.getElementById("but2").onclick=era;
};

function rem(event)
{
var e=event;
var x=e.clientX+window.scrollX;
var y=e.clientY+window.scrollY;
document.getElementById("lay").style.left=x+"px";
document.getElementById("lay").style.top=y+"px";
```

```
document.getElementById("lay").style.visibility="visible";
}

function reta()
{
document.getElementById("lay").style.visibility="hidden";
}

function era()
{
document.getElementById("lay").style.visibility="hidden";
document.getElementById("att").style.display="inline";
window.location="del.php?del="+document.getElementById("sel").value;
}
</script>
</head>

<body>

<div id="lay"><p>Удалить изображение?<br><br>
<input id="but1" type="button" value="  НЕТ  ">    
<input id="but2" type="button" value="   ДА   "></p></div>

<select id="sel"><option selected value="cosmos/ma.jpg">Марс</option>
<option value="cosmos/fo.jpg">Фобос</option>
<option value="cosmos/de.jpg">Деймос</option></select>
<input id="bu" type="button" value="Удалить">

<span id="att"><br><br>Идёт удаление...</span>

</body>
</html>
```

Мы рассмотрели только подготовительную часть удаления файла. Когда клиент нажал кнопку «ДА», управление передается серверной программе, которая выполняет основную работу. Наш пример составлен так, что после удаления фотографии в окно браузера будет загружена новая страница с подтверждением успешного завершения процесса.

Пример можно изменить, сделав удаление снимка в фоновом режиме без перезагрузки страницы. Для этого необходимо прибегнуть к технологии Ajax, о которой мы говорили в третьей главе. Одновременно придется в том же фоновом режиме перезагружать список фотографий, чтобы он соответствовал реальному положению дел на сервере. Как мы уже неоднократно говорили – наши примеры оставляют достаточно места для творческих экспериментов.

Скажем несколько слов о безопасности серверной программы. Так как она принимает адрес файла, это таит в себе определенную опасность. Дело в том, что недоброжелатель может создать собственный запрос к скрипту del.php и ввести в него совершенно иной адрес: например, index.html. В результате будет удалена главная страница вашего сайта. Более того, опытный хакер, хорошо знающий устройство операционной системы компьютера, на котором работает ваш сайт, может, составив соответствующий запрос к del.php, вызвать более серьезные проблемы. Поэтому в реальном файле del.php вы должны предусмотреть такие проверки входящей информации, которые не позволят злоумышленникам навредить вашему сайту и хостингу. Отнеситесь к этому со всей серьезностью!

Завершая рассказ, хотел бы еще добавить, что создавать собственные диалоговые окна вам, наверное, придется еще не раз. Ведь они нужны не только при удалении данных, но во многих других случаях, когда сайт предоставляет посетителю право выбора.

Трансформации

Начать новую тему сразу хотел бы с оговорки. Сам по себе пример из этого раздела вряд ли может иметь какое-то практическое применение. Ценность его в том, что он демонстрирует подходы к созданию визуальных эффектов, применимых в самых разных случаях: в первую очередь, при создании на странице объектов, которые своим поведением должны привлекать особое внимание посетителя. Такими объектами могут быть бегущие строки, рекламные баннеры, заголовки важных объявлений и многие другие.

Создать необходимые эффекты можно как с помощью встроенных в CSS средств трансформации, так и с помощью самостоятельно разработанных функций. Мы попробуем комбинированный вариант.

Зайдите на страницу http://bookjs.ru/v2.html демонстрационного сайта. Вы увидите, как меняется центральный объект. Перед вами квадрат со сторонами 350 пикселей. На ваших глазах он плавно превращается в круг и так же плавно возвращается в первоначальное состояние. Одновременно фигура не менее плавно меняет свой цвет: фиолетовый, красный, синий, зеленый. Оба процесса зациклены, поэтому будут продолжаться до тех пор, пока вы не закроете страницу. Промежуточный этап вы можете видеть на рисунке 5_2.

Наличие подобных объектов на странице обязательно привлечет внимание посетителя. Естественно, что это должны быть элементы, выполняющие определенную информационную или рекламную задачу. Не стоит применять их, пытаясь приукрасить страницу или стараясь поразить воображение клиента.

Трансформациями объекта управляют две функции. В той, что меняет цвет фигуры, использовано свойство CSS transition, регулирующее переходы элементов из одного состояния в другое. Именно это свойство обеспечивает плавность изменения «окраски». Геометрию объекта меняет вторая функция. В ней плавность переходов обеспечивается программными методами.

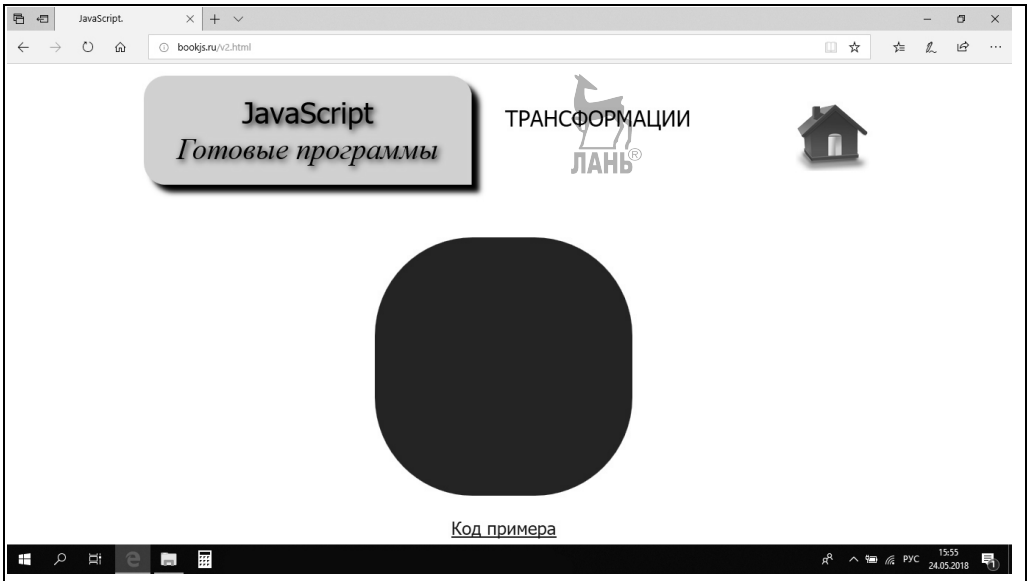

Рис. 5_2. Пример цветовой и геометрической трансформации объекта

Свойство transition *позволяет анимировать изменение свойства какого-либо элемента из одного значения в другое. Поэтому клиент видит в браузере не резкий переход, а плавное преобразование. В самом простом случае* ***transition*** *оперирует двумя значениями: названием свойства элемента и временем изменения этого свойства. Вообще, грамотно манипулируя* ***transition****, программист может добиться очень интересных визуальных эффектов. Обращаю ваше внимание, что* ***transition*** *– свойство CSS. До сих пор в книге подробные комментарии давались только для свойств и методов JavaScript. Тем не менее, в этом случае сделано исключение, так как начинающие программисты редко прибегают к использованию* ***transition****.*

Как всегда, начиная разбирать пример, первым делом разместим на странице необходимые элементы. В данном случае у нас такой элемент один – квадрат. Он образован слоем div:

** **

Настройки слоя:

#lay {width: 350px; height: 350px; background: #660066;}

В исходном состоянии квадрат окрашен в фиолетовый цвет:

background: #660066

Рассмотрим, каким образом можно получить цветовую и геометрическую трансформацию нашей фигуры.

Начнем с изменения цвета. Им управляет функция **tra()** (от transition – переход). Запуск происходит сразу после загрузки страницы:

```
window.addEventListener("load", tra);
```

Перед описанием функции объявим две переменные:

```
var t=0;
var arr=["#CC0000", "#000099", "#006600", "#660066"];
```

Первая является счетчиком цикла, вторая хранит массив с кодами цветов, участвующих в трансформациях.

Итак, страница загрузилась и сразу начинается выполнение функции **tra()**:

```
function tra()
{
document.getElementById("lay").style.transition="background 2s";
document.getElementById("lay").style.background=arr[t];
t++;
if (t==4)
  {
  t=0;
  }
window.setTimeout(tra, 2000);
}
```

Первая инструкция сообщает программе, что анимированию подлежит свойство **background**, продолжительность перехода – 2 секунды. Вторая инструкция сообщает, что переход совершается от исходного состояния фона к значению **arr[t]**. В первом цикле **t=0**, поэтому через 2 секунды фон плавно становится красным (значение **arr[0]** – **#CC0000**). Затем переменная **t** увеличивается на единицу и функция **tra()** вызывается повторно методом **setTimeout()**:

```
window.setTimeout(tra, 2000);
```

Интервал метода 2000 миллисекунд (2 секунды) синхронизирован со временем перехода фона от одного цвета к другому.

На следующем «витке» цикла **t** равно единице, соответственно красный цвет плавно перетекает в синий (значение **arr[1]** – **#000099**). После чего счетчик вновь увеличивается на единицу и принимает значение 2. Следует новый вызов функции **tra()**. Так происходит до тех пор, пока условие **if (t==4)** ложно. Когда оно становится истинным, переменной **t** присваивается начальное значение

```
if (t==4)
  {
```

```
  t=0;
  }
```

и функция «заходит» на следующий круг. Поскольку в программе механизмы остановки отсутствуют, ее выполнение прекращается только после закрытия страницы.

Программа изменения формы квадрата тоже запускается сразу после загрузки страницы:

```
window.addEventListener("load", bor);
```

Но работает она по другому принципу. Если функция **tra()**, образно говоря, накручивает круги, то функция **bor()** (от термина border) двигает процесс вперед-назад по одному «маршруту». Больше всего это напоминает поступательно-возвратные движения поршня. Фигура сначала плавно закругляется, а потом распрямляется обратно. И так непрерывно, пока страница загружена в браузере.

Для реализации такого принципа работы необходимы две переменные:

```
var v=0;
var n=175;
```

Первая является счетчиком пикселей при «закруглении» фигуры, вторая – счетчиком пикселей при «распрямлении».

Функция включает в себя два блока:

```
function bor()
{
if (v<=175)
  {
  document.getElementById("lay").style.borderRadius=v+"px";
  v++;
  window.setTimeout(bor, 10);
  if(v==175)
    {
    n=175;
    }
  }
else
  {
  document.getElementById("lay").style.borderRadius=n+"px";
  n--;
  window.setTimeout(bor, 10);
  if(n==0)
    {
    v=0;
```

```
    }
  }
}
```

В самом начале, после запуска, значение переменной **v** равно 0, поэтому выполняется первый блок инструкций:

```
if (v<=175)
  {
  document.getElementById("lay").style.borderRadius=v+"px";
  v++;
  window.setTimeout(bor, 10);
  if(v==175)
    {
    n=175;
    }
  }
```

Он работает до тех пор, пока **v** меньше 175. Каждые 10 миллисекунд свойству **border-radius** нашей фигуры присваивается новое значение, увеличивающееся на единицу при очередном проходе цикла. За счет этого углы квадрата плавно сглаживаются – до тех пор, пока фигура не превратится в круг. В этот момент выполнение первого блока программы прерывается и начинает работать второй:

```
else
  {
  document.getElementById("lay").style.borderRadius=n+"px";
  n--;
  window.setTimeout(bor, 10);
  if(n==0)
    {
    v=0;
    }
  }
```

Теперь отсчет значений **border-radius** идет в обратном направлении и круг плавно трансформируется в квадрат. При достижении переменной **n** «нулевой отметки» происходит «обнуление» счетчика **v** и опять начинает выполняться первый блок инструкций. Когда **v** достигнет значения 175, переменной **n** будет присвоено такое же значение. Выполнение первого блока инструкций будет остановлено, «в дело» опять вступит второй блок. И так без остановки – до закрытия страницы.

Таким образом, синхронное выполнение функций **tra()** и **bor()** позволяет создать эффект одновременной цветовой и геометрической трансформации.

Листинг с кодом примера приведен ниже:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Трансформации</title>
<style>
#lay {width: 350px; height: 350px; background: #660066;}
</style>

<script>
window.addEventListener("load", tra);
window.addEventListener("load", bor);

var t=0;
var arr=["#CC0000", "#000099", "#006600", "#660066"];

function tra()
{
document.getElementById("lay").style.transition="background 2s";
document.getElementById("lay").style.background=arr[t];
t++;
if (t==4)
  {
  t=0;
  }
window.setTimeout(tra, 2000);
}

var v=0;
var n=175;

function bor()
{
if (v<=175)
  {
  document.getElementById("lay").style.borderRadius=v+"px";
  v++;
  window.setTimeout(bor, 10);
  if(v==175)
    {
    n=175;
    }
  }
```

```
else
  {
  document.getElementById("lay").style.borderRadius=n+"px";
  n--;
  window.setTimeout(bor, 10);
  if(n==0)
    {
    v=0;
    }
  }
}
</script>
</head>

<body>

<div id="lay"></div>

</body>
</html>
```

Мы рассмотрели пример, как программными методами можно акцентировать внимание посетителя на объектах или элементах с нестандартным «поведением». Используя алгоритмы, предложенные в этом разделе, вы можете создавать собственные визуальные эффекты для ваших сайтов. Однако, двигаясь по этому пути, не забывайте о чувстве меры. Времена, когда сверкающие, мигающие, переливающиеся страницы считались «крутыми», давно прошли. Теперь такая «какофония» воспринимается как проявление плохого вкуса. Так что не перебарщивайте.

Для смартфонов и планшетов

Наша книга постепенно приближается к завершению. Осталось всего несколько примеров. Но именно в этот момент мы, наконец, рассмотрим самую простую программу из представленных на демонстрационном сайте. Теоретически, с такой программы надо было бы начинать рассказ о применении JavaScript. Но поскольку она не попадает в предшествующие тематические разделы, то естественным образом оказалась в последней главе вместе с другими «неформатными» примерами.

Ценность нашей программы можно точно охарактеризовать известной пословицей «мал золотник, да дорог». Код примера пригодится тому, кто планирует разрабатывать сайты, которые хорошо смотрятся не только на мониторах компьютеров, но и в различных гаджетах: планшетах и смартфонах. Эта простая программа поможет разработчику адаптировать дизайн сайта под экра-

ны с разным разрешением, а кроме того, менять форматирование при повороте смартфона или планшета из горизонтального положения в вертикальное и наоборот.

Особо хочется обратить внимание на размеры шрифтов. Текст, который хорошо читается при горизонтальном положении смартфона, становится слишком мелким, если повернуть смартфон вертикально. Пример, рассмотренный далее, призван в первую очередь исправить данный недостаток. Но его ценность этим не ограничивается. Добавив в программу необходимые инструкции, управляющие загрузкой контента и его стилевым оформлением, вы можете подстраивать параметры страницы под разные экраны и под разные углы поворота. Мы не станем загружать наш пример демонстрацией неограниченных возможностей метода. Как уже говорилось в начале этой главы, у нас теперь другая задача – продемонстрировать алгоритм создания необходимого кода. Этим мы и займемся.

Что будет делать наша программа? Определять размеры экрана и, в зависимости от этих значений, менять размер шрифта текстового блока. Кроме того, программа будет отслеживать поворот гаджета, подстраивая размер шрифта так, чтобы он легко читался при горизонтальном и вертикальном положении устройства.

В теле документа у нас всего один элемент:

```
<span id="te"></span>
```

Сюда мы будем добавлять информацию, показывающую реакцию программы на различные ситуации: загрузку страницы, а также изменение положения смартфона или планшета.

В головной части документа у нас объявляются два обработчика:

```
window.onload=orien;
window.onorientationchange=orien;
```

Первый вызывает функцию **orien()** сразу после загрузки страницы, второй вызывает ту же функцию в момент поворота мобильного устройства.

Событие orientationchange *характерно для устройств, которые дают возможность пользователю менять ориентацию экрана с альбомной на книжную и обратно. Обычно обработка этого события происходит, когда необходимо изменить внешний вид или содержимое страницы при изменении положения экрана (то есть, подстроить оформление страницы под разные режимы отображения).*

Функция очень простая:

```
function orien()
{
var w=screen.width;
```

```
var h=screen.height;
if(w<600)
  {
  document.getElementById("te").style.fontSize="40px";
  document.getElementById("te").innerHTML="Размеры экрана: "+w+" x
"+h+"<br><br>Размер шрифта: 40px";
  }
else
  {
  document.getElementById("te").style.fontSize="24px";
  document.getElementById("te").innerHTML="Размеры экрана: "+w+" x
"+h+"<br><br>Размер шрифта: 24px";
  }
}
```

В первых двух строках определяются размеры экрана:

```
var w=screen.width;
var h=screen.height;
```

Объект screen *содержит информацию о размерах экрана компьютера или мобильного устройства. Из* ***screen.width*** *разработчик получает ширину экрана, из* ***screen.height*** *– высоту. Свойства* ***screen.availWidth*** *и* ***screen.availHeight*** *возвращают размеры доступной части экрана (то есть, за вычетом элементов интерфейса, таких, как, например, панель задач).*

После чего свойству **font-size** информационной строки присваивается значение, которое зависит от ширины экрана – 40 пикселей или 24 пикселя:

```
document.getElementById("te").style.fontSize="40px";
…
document.getElementById("te").style.fontSize="24px";
```

Затем в информационной строке выводятся размеры экрана и сообщение о том, каким шрифтом отображены эти данные:

```
document.getElementById("te").innerHTML="Размеры экрана: "+w+" x
"+h+"<br><br>Размер шрифта: 40px";
…
document.getElementById("te").innerHTML="Размеры экрана: "+w+" x
"+h+"<br><br>Размер шрифта: 24px";
```

Действующий пример можно посмотреть на странице http://bookjs.ru/v3.html. Лучше всего это делать с помощью смартфона. В горизонтальном положении размер шрифта на странице **24px**. Такой текст легко читается. Это хорошо видно на рисунке 5_3. Если повернуть смартфон вертикально,

изменятся показания в строке «Размеры экрана» и одновременно увеличится кегль шрифта – до 40 пикселей. Поэтому текст по-прежнему будет легко читаемым (рис. 5_4).

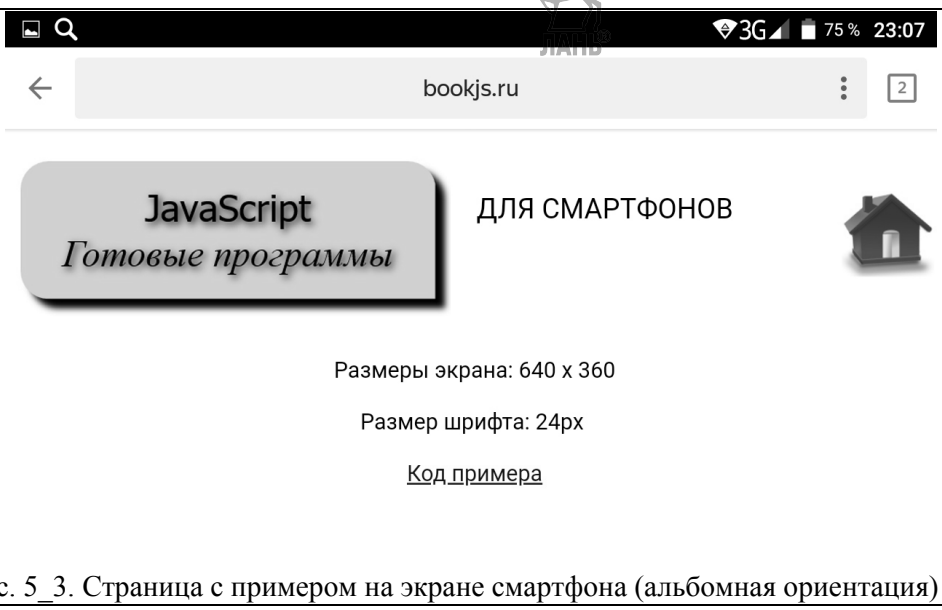

Рис. 5_3. Страница с примером на экране смартфона (альбомная ориентация)

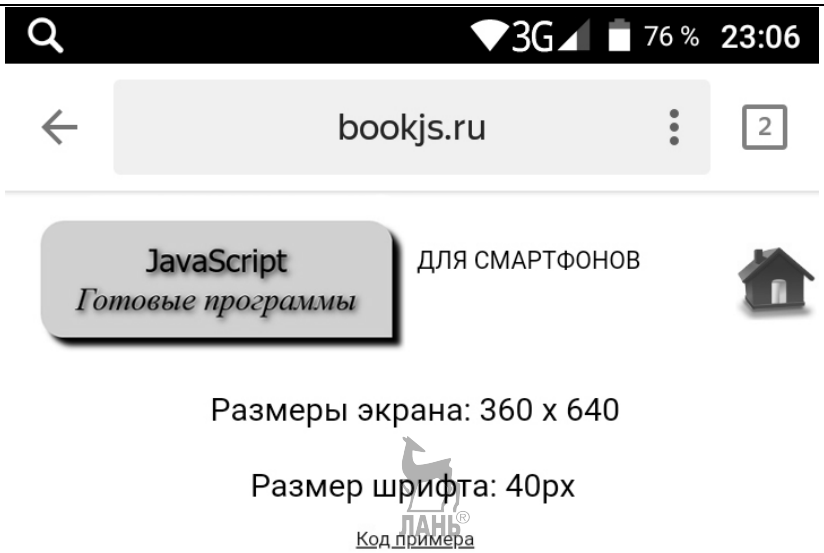

Рис. 5_4. Страница с примером на экране смартфона (книжная ориентация)

Пример целиком:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Для смартфонов и планшетов</title>
<script>

window.onload=orien;
window.onorientationchange=orien;

function orien()
{
var w=screen.width;
var h=screen.height;
if(w<600)
  {
  document.getElementById("te").style.fontSize="40px";
  document.getElementById("te").innerHTML="Размеры экрана: "+w+" x
"+h+"<br><br>Размер шрифта: 40px";
  }
else
  {
  document.getElementById("te").style.fontSize="24px";
  document.getElementById("te").innerHTML="Размеры экрана: "+w+" x
"+h+"<br><br>Размер шрифта: 24px";
  }
}
</script>
</head>

<body>

<span id="te"></span>

</body>
</html>
```

Повторюсь: определив размеры экрана, вы можете как угодно переформатировать отдельные элементы или даже всю страницу, подстраивая их под текущее разрешение монитора. Более того, применив технологию Ajax, можно вообще полностью менять контент, загружая разные варианты страницы, написанные для разных устройств.

Продемонстрируем такой способ на конкретном примере. Его вы можете посмотреть на странице http://bookjs.ru/v3_2.html.

Пусть у нас будет документ, в который при смене ориентации мобильного устройства станут загружаться разные изображения. Главный элемент страницы – слой для загрузки картинок:

```
<span id="pas"></span>
```

У нас будет 2 обработчика событий:

```
window.onload=pla;
window.onorientationchange=pla;
```

Оба вызывают одну и ту же функцию после загрузки документа, а также при смене ориентации смартфона или планшета.

Инициируем переменную, которая меняет свое значение в зависимости от ориентации устройства:

```
var n=1;
```

Единица соответствует вертикальному положению смартфона или планшета, двойка – горизонтальному.

Напишем функцию, обрабатывающую изменение ориентации:

```
function pla()
{
var w=screen.width;
if(w<600)
  {
  n=1;
  }
else
  {
  n=2;
  }
var re=new XMLHttpRequest();
re.onload=pres;
var adr="paste.php?nam="+n;
re.open("GET", adr, true);
re.send();
}
```

Сначала мы определяем, как изменилась ширина экрана, а затем, в зависимости от его размера, с помощью технологии Ajax отсылаем на сервер запрос,

в котором указан адрес скрипта и параметр ориентации. В ответ скрипт **paste.php** возвращает нам соответствующее изображение:

```
function pres()
{
var resp=this.responseText;
document.getElementById("pas").innerHTML=resp;
}
```

Помещаем HTML-код, полученный в качестве ответа, в слой **pas**. Если наше мобильное устройство было в вертикальном положении, мы увидим картинку, как на рисунке 5_5. Если в горизонтальном, картинка изменится (см. рис. 5_6).

Рис. 5_5. Загрузка контента при вертикальном положении смартфона

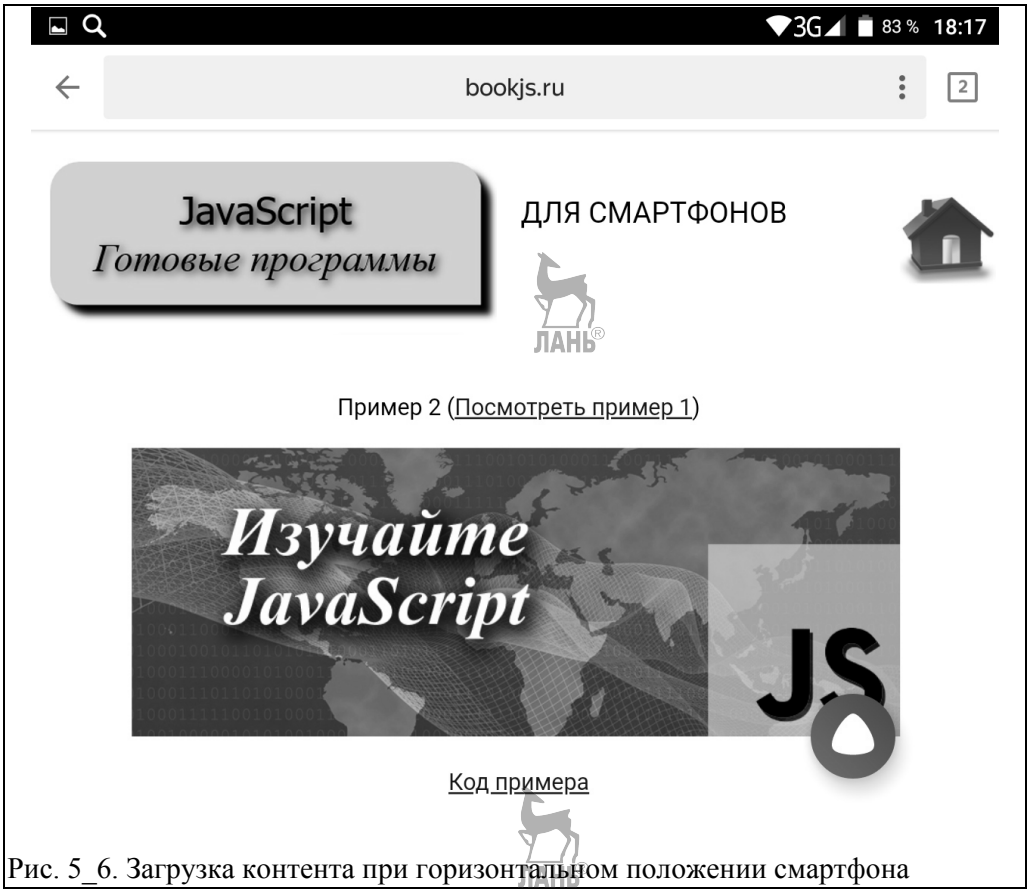

Рис. 5_6. Загрузка контента при горизонтальном положении смартфона

Код примера так же прост, как и в предыдущем случае:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>JavaScript. Полезные рецепты. Для смартфонов и планшетов</title>
<style>
a {color: #0000CC;}
a:hover {color: #CC0000;}
</style>

<script>
window.onload=pla;
window.onorientationchange=pla;

var n=1;
```

```
function pla()
{
var w=screen.width;
if(w<600)
  {
  n=1;
  }
else
  {
  n=2;
  }
var re=new XMLHttpRequest();
re.onload=pres;
var adr="paste.php?nam="+n;
re.open("GET", adr, true);
re.send();
}

function pres()
{
var resp=this.responseText;
document.getElementById("pas").innerHTML=resp;
}
</script>
</head>

<body>

<span id="pas"></span>

</div>

</body>
</html>

---------- КОД СКРИПТА paste.php ----------

<?php
$i=$_REQUEST["nam"];

if(preg_match('/[^12]/', $i))
{
echo "НЕКОРРЕКТНЫЕ ДАННЫЕ !<br><br>";
}
else
```

```
{
echo "<img src=\"pict/or$i.jpg\" alt=\"Фото\">";
}
```

Полагаю, двух примеров достаточно, чтобы читатель убедился: управлять содержимым страницы, предназначенной для просмотра на мобильном устройстве, легко и просто. Минимум кода – и вы получите желаемый результат.

Бесконечная лента новостей

В последнем разделе первой главы мы обсуждали ситуацию с загрузкой данных по частям. Тогда наши примеры были рассчитаны на добавление ограниченного объема контента. Он сразу размещался в скрытой форме на странице, а затем добавлялся в видимую область документа небольшими порциями. Принцип был хорош для случаев, когда информации немного и разработчик сразу знает, сколько ее. Но нередко программист стоит перед задачей по частям добавлять заранее неизвестный объем текста и фотографий (и во многих случаях – очень большой). Типичный пример – социальные сети. В главной ленте новостей, в группах, в личных и корпоративных аккаунтах сообщения и фотографии загружаются по несколько штук по мере прокрутки страницы в окне браузера. Причем, количество дозагрузок ограничено только объемом файла или базы данных, из которых черпается информация. В результате, постоянно смещая вниз движок полосы прокрутки, вы можете загрузить на страницу и 100, и 1000 и много больше сообщений.

Для реализации подобных методов необходимы специальные программы. Мы с вами уже получили достаточный опыт создания разнообразных приложений на JavaScript. Давайте теперь попробуем сделать собственную версию бесконечной ленты новостей.

У нас будет страница с определенным, предустановленным содержимым, достаточно большим, чтобы не умещаться в окне браузера на мониторе с самым высоким разрешением. Пример такой страницы – на демонстрационном сайте: http://bookjs.ru/v4.html. На ней в качестве «изначального» контента – арабские цифры и буквы английского алфавита, выстроенные вертикально для создания необходимой «длины» документа. Скриншот данной страницы я решил не делать, так как он не сможет передать эффект добавления информации.

Поэкспериментируем. Прокрутите страницу вниз до упора. Ползунок неожиданно укоротится и немного сместится вверх, а на странице появится несколько одинаковых фотографий, озаглавленных «КОНТЕНТ № 1», «КОНТЕНТ № 2», «КОНТЕНТ № 3» и так далее. Попробуйте опять прокрутить страницу вниз до упора. Эффект повторится: ползунок еще укоротится и вновь сместится вверх, а на странице появится очередная порция снимков. Так будет происходить до тех пор, пока вы не решите, что экспериментов достаточно. Это и есть наша версия бесконечной ленты новостей. Почти одинаковый текст и совершенно одинаковые фотографии использованы, чтобы упростить реализацию

нашего примера (конечно, сотни или тысячи разных картинок сделают пример более натуральным, но никак не изменят его сути, поэтому автор не стал усложнять себе жизнь).

Кликните на ссылке «Код примера», чтобы посмотреть, как устроена наша лента. Контент страницы, как мы уже говорили, образован из арабских цифр и букв английского алфавита, расположенных пирамидой:

```
<br><br>0<br><br>1<br><br>2<br><br>3<br><br>4<br><br>5
<br><br>6<br><br>7<br><br>8<br><br>9<br><br>----------
<br><br>a<br><br>b<br><br>c<br><br>d<br><br>e<br><br>f
<br><br>g<br><br>h<br><br>i<br><br>j<br><br>k<br><br>l
<br><br>m<br><br>n<br><br>o<br><br>p<br><br>q<br><br>r
<br><br>s<br><br>t<br><br>u<br><br>v<br><br>w<br><br>x
<br><br>y<br><br>z<br><br>
```

Ниже имеется область, предназначенная для добавления текста и фотографий:

```
<span id="ins"></span>
```

Наша программа должна работать во время прокрутки страницы, поэтому зарегистрируем соответствующий обработчик события:

```
window.onscroll=scro;
```

Перед описанием функции **scro()** создадим переменную-счетчик:

```
var i=1;
```

В нашем примере она не играет существенной роли. Дело в том, что переменная **i** нужна только для нумерации порций данных, поступающих с сервера. В реальном проекте с помощью такой переменной вы можете передавать на сервер порядковый номер блока информации, который нужно отобразить на странице. Более того, переменных может быть несколько, если возвращаемое содержимое определяется несколькими параметрами или условиями.

Теперь сама функция:

```
function scro()
{
if(window.pageYOffset+window.innerHeight>=document.body.clientHeight)
  {
  var re=new XMLHttpRequest();
  re.onload=demo;
  var adr="insert.php?ind="+i;
  re.open("GET", adr, true);
```

```
  re.send();
  i++;
  }
}
```

Все самое главное прописано в первой строке:

```
if(window.pageYOffset+window.innerHeight>=document.body.clientHeight)
  {
  …
  }
```

Мы берем величину прокрутки страницы

window.pageYOffset

складываем с внутренним размером окна браузера

window.innerHeight

и сравниваем полученный результат с «высотой» документа

document.body.clientHeight

Свойство pageYOffset *содержит информацию о величине вертикальной прокрутки текущей страницы в пикселях.*
Прежде, чем давать определения ***innerHeight*** *и* ***clientHeight****, должен заметить, что в разных источниках, у разных авторов, на разных web-ресурсах я встречал очень разные, иногда взаимоисключающие описания этих свойств. Не желая участвовать в теоретических спорах, я использовал характеристики, которые дает этим свойствам Mozilla Foundation на сайте developer.mozilla.org. Применив* ***innerHeight*** *и* ***clientHeight*** *в контексте, предложенном этой уважаемой организацией, мы получили в нашем примере требуемый результат.*
Свойство innerHeight *содержит информацию о внутренней высоте окна браузера в пикселях (включая полосу горизонтальной прокрутки, если она есть).*
Свойство clientHeight *содержит информацию о внутренней высоте элемента или страницы в пикселях (без учета размеров границы и внешних отступов).*

До тех пор, пока страница прокручена не до конца, условие

window.pageYOffset+window.innerHeight>=document.body.clientHeight

ложно (точнее, почти до конца; тело нашей страницы имеет небольшие отступы от верхнего и нижнего края окна браузера, которые создаются по умолчанию, когда эти свойства не определены в настройках страницы; в результате реальная величина полной прокрутки у нас несколько больше, чем «высота страницы»; поэтому равенство в выражении достигается «за несколько пикселей» до конца прокрутки). Но

как только документ окажется прокручен полностью (точнее, почти полностью), условие станет истинным и будет выполнен следующий блок инструкций:

```
var re=new XMLHttpRequest();
re.onload=demo;
var adr="insert.php?ind="+i;
re.open("GET", adr, true);
re.send();
i++;
```

Это не что иное, как уже хорошо знакомая нам отправка данных на сервер с использованием технологии Ajax. Мы посылаем запрос, передавая скрипту **insert.php** в качестве параметра значение переменной **i**. Увеличение переменной на единицу в конце функции необходимо, чтобы в следующем запросе имитировать передачу параметра с новым значением.

Скрипт **insert.php**, расположенный на сервере, очень простой: он отправляет в качестве ответа текст «КОНТЕНТ №», добавляя к значку номера значение, переданное в запросе, и фотографию планшетного компьютера, которая нам уже встречалась в других примерах. Расширение **php** указывает, что скрипт написан на PHP (все серверные программы из пятой главы написаны на этом языке, как и было обещано в предисловии).

Данные, возвращенные с сервера, получает функция **demo()**:

```
function demo()
{
var rep=this.responseText;
document.getElementById("ins").insertAdjacentHTML("beforeend", rep);
}
```

Новые изображение и текст каждый раз добавляются в конец контейнера **ins**.

Вот так незатейливо работает метод бесконечной загрузки данных.

Соберем все «компоненты» нашей программы. У нас получится простой и короткий листинг:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Бесконечная лента</title>
<script>
window.onscroll=scro;

var i=1;
function scro()
```

```
{
if(window.pageYOffset+window.innerHeight>=document.body.clientHeight)
  {
  var re=new XMLHttpRequest();
  re.onload=demo;
  var adr="insert.php?ind="+i;
  re.open("GET", adr, true);
  re.send();
  i++;
  }
}

function demo()
{
var rep=this.responseText;
document.getElementById("ins").insertAdjacentHTML("beforeend", rep);
}
</script>
</head>

<body>

<br><br>0<br><br>1<br><br>2<br><br>3<br><br>4<br><br>5
<br><br>6<br><br>7<br><br>8<br><br>9<br><br>----------
<br><br>a<br><br>b<br><br>c<br><br>d<br><br>e<br><br>f
<br><br>g<br><br>h<br><br>i<br><br>j<br><br>k<br><br>l
<br><br>m<br><br>n<br><br>o<br><br>p<br><br>q<br><br>r
<br><br>s<br><br>t<br><br>u<br><br>v<br><br>w<br><br>x
<br><br>y<br><br>z<br><br>

<span id="ins"></span>

</body>
</html>

---------- КОД СКРИПТА insert.php ----------

<?php
$i=$_REQUEST["ind"];

if(preg_match('/[^0-9]/', $i))
{
echo "НЕКОРРЕКТНЫЕ ДАННЫЕ !<br><br>";
}
else
```

```
{
echo "КОНТЕНТ № $i:<br><img src=\"pict/comp.jpg\" alt=\"Фото\"><br><br>";
}
```

Регулярно наблюдаю на различных программистских форумах обсуждение тем из разряда «как создать бесконечную ленту новостей». Довольно часто авторы сообщений предлагают на удивление «замороченные» способы. Наш пример демонстрирует, что на самом деле решение у этой задачи очень простое.

Калькулятор

До сих пор мы вели речь о создании кода, который играл вспомогательную роль. У нас были web-страницы, нацеленные на выполнение разных задач, а функции JavaScript служили инструментами для их реализации. Настало время поговорить о случаях, когда программы, написанные на JavaScript, являются совершенно самостоятельными приложениями. Один из таких случаев – калькулятор.

Сразу предупрежу – это довольно сложная программа (сложнее только пример из последнего раздела пятой главы). Мы разберем принцип работы калькулятора, объясним назначение функций программы, но полный листинг приводить не станем. Он слишком длинный, займет много места, поэтому желающие увидеть код страницы целиком могут зайти на сайт поддержки: http://bookjs.ru/calc/calc.html. Попробуйте инструмент в действии, а потом пройдите по ссылке «Код примера», чтобы посмотреть «первоисточник».

Как видите, наш калькулятор представляет собой упрощенный вариант подобных устройств или программ (рис. 5_7). Он выполняет четыре основных арифметических действия: сложение, вычитание, умножение и деление. Программа оперирует целыми числами и десятичными дробями с положительными или отрицательными значениями. Впрочем, сами вычисления делает JavaScript, а созданные нами функции просто запускают соответствующие процессы.

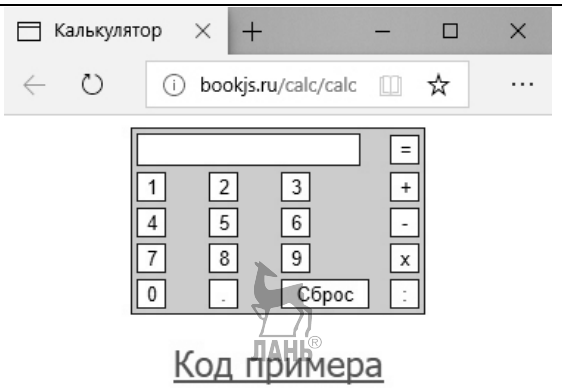

Рис. 5_7. Калькулятор на JavaScript

Как всегда, первым делом создаем необходимый интерфейс:

```
<table>
<tr>
<td colspan="3">
<input type="hidden" id="SKR" value="0">
<input type="hidden" id="DAT" value="0">
<input type="hidden" id="DEI" value="+">
<input type="text" readonly id="SUM" class="t1"></td>
<td><input id="ra" type="button" value="=" class="b1"></td>
</tr>
<tr>
<td class="wi1"><input id="s1" type="button" value="1" class="b1"></td>
<td class="wi1"><input id="s2" type="button" value="2" class="b1"></td>
<td class="wi2"><input id="s3" type="button" value="3" class="b1"></td>
<td><input id="su" type="button" value="+" class="b1"></td>
</tr>
<tr>
<td><input id="s4" type="button" value="4" class="b1"></td>
<td><input id="s5" type="button" value="5" class="b1"></td>
<td><input id="s6" type="button" value="6" class="b1"></td>
<td><input id="mi" type="button" value="-" class="b1"></td>
</tr>
<tr>
<td><input id="s7" type="button" value="7" class="b1"></td>
<td><input id="s8" type="button" value="8" class="b1"></td>
<td><input id="s9" type="button" value="9" class="b1"></td>
<td><input id="um" type="button" value="x" class="b1"></td>
</tr>
<tr>
<td><input id="s0" type="button" value="0" class="b1"></td>
<td><input id="to" type="button" value="." class="b1"></td>
<td><input id="sb" type="button" value="Сброс" class="b2"></td>
<td><input id="de" type="button" value=":" class="b1"></td>
</tr>
</table>
```

Табло и кнопки калькулятора мы разместили в теле таблицы. Ее настройки:

```
table {width: 200px; border: 1px solid #0000CC;
background: #CCCCCC; margin: 0px auto; text-align: left;}
```

Располагаем калькулятор по центру страницы:

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

Устанавливаем свойства кнопок:

```
.b1 {width: 20px; height: 20px; background: #FFFFFF; border: 1px solid #0000CC;}
.b2 {width: 60px; height: 20px; background: #FFFFFF; border: 1px solid #0000CC;}
.wi1 {width: 45px;}
.wi2 {width: 70px;}
```

Добавляем настройки табло:

```
.t1 {width: 150px; height: 18px; border: 1px solid #0000CC;
font-family: Tahoma; font-size: 10pt; color:#000000;}
```

Теперь необходимо дать пару комментариев нашим действиям.

1. На странице есть три скрытых поля:

```
<input type="hidden" id="SKR" value="0">
<input type="hidden" id="DAT" value="0">
<input type="hidden" id="DEI" value="+">
```

Они необходимы для хранения промежуточных результатов вычислений. Их назначение объясним, комментируя работу программы.

2. В тег табло мы добавили атрибут **readonly**:

```
<input type="text" readonly id="SUM" class="t1">
```

Он запрещает пользователю самостоятельно добавлять какие-либо данные в текстовое поле. В нем отображаются только вводимые значения и результаты вычислений.

Зарегистрируем обработчики событий. Поскольку их много, мы применим сокращенную форму записи (в одну строку):

```
window.onload=function()
{
document.getElementById("ra").onclick=function(){ravn();};
document.getElementById("su").onclick=function(){rez('+');};
document.getElementById("mi").onclick=function(){rez('-');};
document.getElementById("um").onclick=function(){rez('x');};
document.getElementById("de").onclick=function(){rez('d');};
document.getElementById("s1").onclick=function(){slag(1);};
document.getElementById("s2").onclick=function(){slag(2);};
document.getElementById("s3").onclick=function(){slag(3);};
```

```
document.getElementById("s4").onclick=function(){slag(4);};
document.getElementById("s5").onclick=function(){slag(5);};
document.getElementById("s6").onclick=function(){slag(6);};
document.getElementById("s7").onclick=function(){slag(7);};
document.getElementById("s8").onclick=function(){slag(8);};
document.getElementById("s9").onclick=function(){slag(9);};
document.getElementById("s0").onclick=function(){slag(0);};
document.getElementById("to").onclick=function(){slag('.');};
document.getElementById("sb").onclick=function(){sbros();};
};
```

Начнем с самой простой функции **sbros()**, которая запускается нажатием кнопки «Сброс»:

```
function sbros()
{
document.getElementById('SUM').value='';
document.getElementById('SKR').value='0';
document.getElementById('DAT').value='0';
document.getElementById('DEI').value='+';
}
```

Назначение функции – очистить табло калькулятора от предыдущих вычислений и установить свойствам **value** скрытых полей исходные значения.

Перед описанием остальных функций сначала объявим переменную, которая будет выполнять роль идентификатора процесса:

```
var pere=0;
```

Нажатие кнопок с цифрами или точкой (отделяющей целую часть числа от дробной) запускает функцию **slag()**. Поскольку она первая вступает в действие, с нее и начнем описание принципа работы калькулятора:

```
function slag(A)
{
if(document.getElementById("DAT").value==1)
  {
  document.getElementById("SUM").value="";
  document.getElementById("DAT").value=0;
  }
if((A==".")&&(""==document.getElementById("SUM").value))
  {
  document.getElementById("SUM").value=0+A;
  }
else
```

```
  {
  var znach=document.getElementById("SUM").value;
  document.getElementById("SUM").value=znach+A;
  }
pere=0;
}
```

Объяснять логику работы функции лучше на конкретном примере. Представим, что мы хотим узнать сумму чисел 10.5 и 2. Вы запустили калькулятор и нажали кнопку с цифрой «1». Обработчик события

```
document.getElementById("s1").onclick=function(){slag(1)};
```

передает в функцию аргумент со значением 1. Первый и второй блоки пропускаются, так как исходное значение в поле **DAT** равно 0 и нажата клавиша «1», а не ".". Первая инструкция третьего блока присваивает переменной **znach** содержимое табло. В начальный момент оно пустое, поэтому следующая инструкция

```
document.getElementById("SUM").value=znach+A;
```

отобразит в текстовом поле цифру 1. Теперь нажмем кнопку «0». Первые два блока функции опять будут пропущены, в переменной **znach** окажется цифра 1, а вторая инструкция добавит к ней цифру 0. В результате на табло появится число 10. Затем нажмем точку и следом – цифру «5». На табло появится первое слагаемое 10.5.

Прежде чем продолжать действия, объясним назначение остальных двух блоков. Инструкции

```
if(document.getElementById("DAT").value==1)
  {
  document.getElementById("SUM").value="";
  document.getElementById("DAT").value=0;
  }
```

необходимы, чтобы удалить из табло первое слагаемое, когда вы начинаете вводить второе, и установить исходное значение в поле **DAT**. Блок

```
if((A==".")&&(""==document.getElementById("SUM").value))
  {
  document.getElementById("SUM").value=0+A;
  }
```

нужен на тот случай, если пользователь хочет ввести дробное число, начинающееся с нуля, но забывает нажать кнопку «0», а сразу кликает на точке. Тогда

цифру 0 за клиента добавляет программа (вообще-то, какую бы цифру клиент ни забыл поставить перед точкой, программа добавляет 0; но это происходит только в одном случае – если первая нажатая кнопка – ".").

Итак, первое число введено. Жмем кнопку со знаком «плюс». Происходит запуск функции **rez()**, в которую в качестве аргумента передается символ «+»:

```
function rez(B)
{
var itog="";
if((B=="-")&&(pere==1))
  {
  document.getElementById("SUM").value="-";
  document.getElementById("DAT").value=0;
  pere=0;
  }
if((document.getElementById("SUM").value.indexOf("-
")==0)&&(document.getElementById("SUM").value.length>=2))
  {
  document.getElementById("SUM").value=parseFloat(document.get
ElementById("SUM").value);
  }
if(""!=document.getElementById("SUM").value)&&("-
"!=document.getElementById("SUM").value)&&(pere!=1))
  {
  if(document.getElementById("DEI").value=="+")
    {

itog=parseFloat(document.getElementById("SKR").value)+parseFloat(docume
nt.getElementById("SUM").value);
    }
  if(document.getElementById("DEI").value=="-")
    {
    itog=parseFloat(document.getElementById("SKR").value)-
parseFloat(document.getElementById("SUM").value);
    }
  if(document.getElementById("DEI").value=="x")
    {

itog=parseFloat(document.getElementById("SKR").value)*parseFloat(docume
nt.getElementById("SUM").value);
    }
  if(document.getElementById("DEI").value=="d")
    {
```

```
itog=parseFloat(document.getElementById("SKR").value)/parseFloat(documen
t.getElementById("SUM").value);
    }
  document.getElementById("SKR").value=itog;
  document.getElementById("SUM").value=itog;
  document.getElementById("DAT").value=1;
  pere=1;
  }
if(""!=document.getElementById("SUM").value)&&("-
"!=document.getElementById("SUM").value))
  {
  document.getElementById("DEI").value=B;
  }
if((document.getElementById("SUM").value=="")&&(B=="-"))
  {
  document.getElementById("SUM").value="-";
  }
}
```

Первым делом создадим переменную **itog**, в которую станем помещать промежуточные результаты вычислений:

```
var itog="";
```

Дальше нарушим порядок и начнем разбор функции сразу с последнего блока:

```
if((document.getElementById("SUM").value=="")&&(B=="-"))
  {
  document.getElementById("SUM").value="-";
  }
```

Он необходим на тот случай, когда при пустом табло была нажата кнопка «–». Эта ситуация определяется, как желание клиента ввести в качестве первого значения отрицательное число, а не как указание сделать вычитание.

Вернемся в начало и рассмотрим второй блок. В нем выполняется проверка на наличие знака минус и цифр в поле табло. Если они есть, то этот набор символов преобразуется в отрицательное число:

```
if((document.getElementById("SUM").value.indexOf("-
")==0)&&(document.getElementById("SUM").value.length>=2))
  {
```

```
  document.getElementById("SUM").value=parseFloat(document.get
ElementById("SUM").value);
  }
```

Функция parseFloat *преобразует строку цифр в число. Функция способна различать не только целые числа, но и десятичные дроби, не только положительные, но и отрицательные значения. Одним словом, функция оперирует с вещественными числами. Целая часть должна быть отделена от дробной точкой.*

Мы ввели в качестве первого слагаемого положительное число, поэтому в нашем случае сразу выполняется третий блок:

```
if((""!=document.getElementById("SUM").value)&&("-
"!=document.getElementById("SUM").value)&&(pere!=1))
  {
  …
  }
```

В нем четыре похожие по структуре и назначению инструкции:

```
if(document.getElementById("DEI").value=="+")
  {
  itog=parseFloat(document.getElementById("SKR").value)+parseFloat
(document.getElementById("SUM").value);
  }
if(document.getElementById("DEI").value=="-")
  {
  itog=parseFloat(document.getElementById("SKR").value)-parseFloat
(document.getElementById("SUM").value);
  }
if(document.getElementById("DEI").value=="x")
  {

itog=parseFloat(document.getElementById("SKR").value)*parseFloat(document.
getElementById("SUM").value);
  }
if(document.getElementById("DEI").value=="d")
  {

itog=parseFloat(document.getElementById("SKR").value)/parseFloat(documen
t.getElementById("SUM").value);
  }
```

Обратите внимание: так как в исходном состоянии полю **DEI** присвоено значение «+», то после ввода первого числа нажатие любой из кнопок арифметического действия приведет к выполнению одной и той инструкции:

```
if(document.getElementById("DEI").value=="+")
  {

itog=parseFloat(document.getElementById("SKR").value)+parseFloat(docume
nt.getElementById("SUM").value);
  }
```

Значение поля **SKR** – 0, поэтому сумма чисел из скрытого поля и табло будет равняться 10.5.

Полученный результат присваивается полю табло **SUM** и скрытому полю **SKR**:

```
document.getElementById("SKR").value=itog;
document.getElementById("SUM").value=itog;
```

Последние два оператора в блоке устанавливают значения идентификаторов, роль которых выполняет скрытое поле **DAT** и переменная **pere**:

```
document.getElementById("DAT").value=1;
pere=1;
```

Теперь рассмотрим предпоследний блок

```
if((""!=document.getElementById("SUM").value)&&("-
"!=document.getElementById("SUM").value))
  {
  document.getElementById("DEI").value=B;
  }
```

Здесь выполняется проверка: есть ли уже какие-то данные в поле табло и не состоят ли эти данные из единственного символа – знака «минус»? Мы ввели число 10.5, а значит оно благополучно преодолеет данный «фильтр». В результате скрытое поле **DEI** примет значение, переданное в функцию в качестве аргумента. В нашем случае это символ «+».

Итак, что мы имеем на промежуточном этапе:

– на табло у нас первое введенное число 10.5;

– это число также хранится в скрытом поле **SKR**;

– скрытое поле **DEI** «запомнило» действие, которое предстоит выполнить – сложение.

Нажмем кнопку с цифрой «2» – это наше второе слагаемое. Вновь запускается функция **slag()**. Сначала будет выполнен первый блок:

```
if(document.getElementById("DAT").value==1)
  {
  document.getElementById("SUM").value="";
  …
  }
```

Экран очистится, после чего инструкции из третьего блока

```
var znach=document.getElementById("SUM").value;
document.getElementById("SUM").value=znach+A;
```

выведут на табло второе слагаемое. Осталось нажать кнопку «=», чтобы узнать результат. Эта кнопка запустит функцию **ravn()**:

```
function ravn()
{
var fin="";
if(""!=document.getElementById("SUM").value)&&(""!=document.get
ElementById("SKR").value)&&("-"!=document.getElementById
("SUM").value)&&(pere!=1))
  {
  if(document.getElementById("DEI").value=="+")
    {

fin=parseFloat(parseFloat(document.getElementById("SKR").value)+parseFloa
t(document.getElementById("SUM").value));
    }
  if(document.getElementById("DEI").value=="-")
    {
    fin=parseFloat(parseFloat(document.getElementById("SKR").value)-
parseFloat(document.getElementById("SUM").value));
    }
  if(document.getElementById("DEI").value=="x")
    {

fin=parseFloat(parseFloat(document.getElementById("SKR").value)*parseFloa
t(document.getElementById("SUM").value));
    }
  if(document.getElementById("DEI").value=="d")
    {

fin=parseFloat(parseFloat(document.getElementById("SKR").value)/parseFloa
t(document.getElementById("SUM").value));
    }
  document.getElementById("SKR").value=fin;
```

```
  document.getElementById("SUM").value=fin;
  document.getElementById("DAT").value=1;
  document.getElementById("DEI").value="+";
  pere=1;
  }
}
```

Переменная **fin** нужна для сохранения результата вычислений.

Как и в предыдущей функции, здесь четыре блока похожих инструкций. Сначала на основе данных из поля **DEI** выполняется проверка, какое арифметическое действие необходимо произвести. Затем создается выражение, в котором первым записывается значение из скрытого поля **SKR**, потом символ действия и, наконец, второе значение, взятое из поля **SUM**. Выражение вычисляется и его результат записывается в переменную **fin**. В нашем случае, 10.5 суммируется с 2 и получается 12.5. Инструкция

```
document.getElementById("SUM").value=fin;
```

помещает результат в поле табло. Инструкция

```
document.getElementById("SKR").value=fin;
```

записывает результат в скрытое поле **SKR**. Это делается на случай, если пользователь захочет выполнить еще какие-то действия с результатом. Последние три оператора устанавливают необходимые значения в скрытых полях и глобальной переменной **pere**:

```
document.getElementById("DAT").value=1;
document.getElementById("DEI").value="+";
pere=1;
```

Если вы внимательно прочитали описание принципа действия калькулятора, то должны были обратить внимание на одну особенность функции **rez()**. В ней есть четыре инструкции:

```
itog=parseFloat(document.getElementById("SKR").value)+parseFloat(document.
getElementById("SUM").value);
…
itog=parseFloat(document.getElementById("SKR").value)-
parseFloat(document.getElementById("SUM").value);
…
itog=parseFloat(document.getElementById("SKR").value)*parseFloat(document.
getElementById("SUM").value);
…
```

```
itog=parseFloat(document.getElementById("SKR").value)/parseFloat(document.
getElementById("SUM").value);
```

Однако, разбирая наш пример, мы говорили: так как в исходном состоянии полю **DEI** присвоено значение «+», то после ввода первого числа нажатие любой из кнопок арифметического действия приведет к выполнению только первой инструкции. Спрашивается, для чего тогда нужны остальные три, да к тому же дублирующие аналогичные возможности функции **ravn()**?

Дело в том, что функция **ravn()** подводит финальную черту под вычислениями. А если бы наш пример включал не два слагаемых и одно действие, а больше? Например, мы бы захотели вычислить выражение (10.5+2–3.5)*3? Вот тогда нам и понадобятся «лишние» инструкции из функции **rez()**, которые выполнят все промежуточные вычисления, а подвести итог предоставят функции **ravn()**. В этом случае процесс выглядел бы следующим образом. На первом этапе на табло у нас будет число 10.5, оно же окажется в скрытом поле **SKR**, а в скрытом поле **DEI** сохранится символ сложения. После введения числа 2 мы нажмем кнопку не со знаком «равно», а со знаком «минус». Будет вычислено выражение 10.5+2. Результат 12.5 появится на табло и в скрытом поле **SKR**, а в скрытое поле **DEI** запишется символ вычитания. Введем число 3.5 и нажмем кнопку со знаком умножения. Так как теперь верно условие

```
document.getElementById("DEI").value=="-"
```

то выполнится инструкция

```
itog=parseFloat(document.getElementById("SKR").value)-
parseFloat(document.getElementById("SUM").value);
```

функции **rez()**. Из 12.5 будет вычтено 3.5, результат 9 появится на табло и в скрытом поле **SKR**, а в скрытом поле **DEI** окажется знак умножения. Наконец, нажатием кнопки «=» запустим функцию **ravn()** и получим окончательный ответ: 27. Таким образом, мы убедились, что три незадействованные в первом вычислении инструкции начинают работать, когда наши расчеты состоят из нескольких этапов и с использованием разных арифметических действий.

Необходимо также объяснить наличие блока

```
if((B=="-")&&(pere==1))
  {
  document.getElementById("SUM").value="-";
  document.getElementById("DAT").value=0;
  pere=0;
  }
```

в функции **rez()**. Он необходим, чтобы определить момент, когда, произведя часть вычислений, пользователь хочет ввести не знак операции вычитания, а

отрицательное число. Подчеркнем, этот блок действует только на промежуточных этапах вычислений.

Подведем итог. Наш калькулятор позволяет делать несложные вычисления, а также корректно исправляет ошибки пользователя. А еще является совершенно самостоятельной программой, которую можно разместить на сайте или на своем компьютере.

Игра «Сбей НЛО»

Изучая различные книги по программированию, я обратил внимание на одну тенденцию: если авторы рассказывают о создании игр на том или ином языке, то чаще всего в качестве примера берут логические игры. Мы, для разнообразия, нарушим эту традицию и рассмотрим создание простейшего космического симулятора. Геймер в такой игре будет охотится на НЛО, который скачкообразно перемещается из одной точки в другую (скорее всего, экипаж летающей тарелки использует моментальный переход в гиперпространстве). У пилота земного корабля в запасе 15 зарядов для поражения НЛО. Каждое попадание наносит звездолету инопланетян урон в размере 20% от его первоначального уровня живучести. Пять точных попаданий – и противник сбит. Мазали чаще, чем дозволено правилами, значит проигравшая сторона – вы.

Игру в действии можно увидеть на странице http://bookjs.ru/nlo/nlo.html демонстрационного сайта. Если в данный момент вы не подключены к Интернету, то можете познакомиться с интерфейсом симулятора, посмотрев на рисунок 5_8.

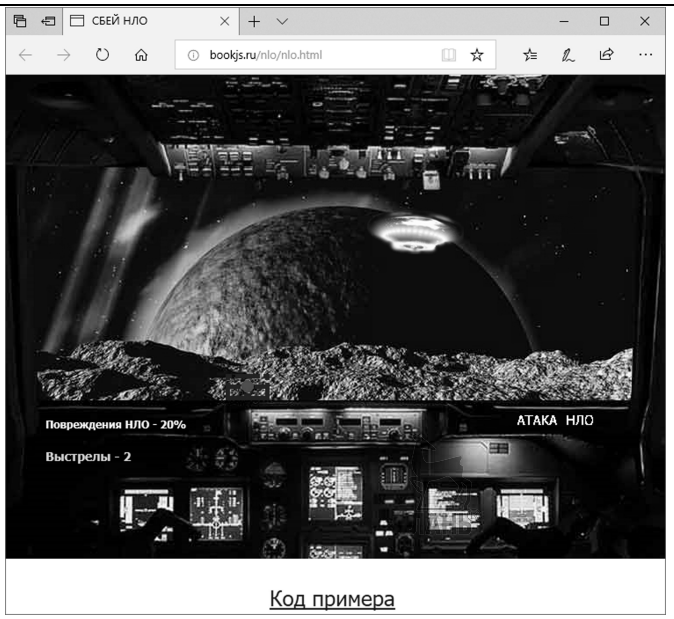

Рис. 5_8. Игра «Сбей НЛО»

В качестве графической основы игры использован рисунок с одного из стоков бесплатных фотографий. Он изображает внутренности космического корабля. Через центральный экран вы наблюдаете планетарный ландшафт, на фоне которого разворачивается поединок. «Пилот» видит перемещающийся НЛО и прицел, управлять которым может, двигая компьютерную мышь вправо-влево. Чтобы произвести выстрел, необходимо нажать левую клавишу мыши. Вслед за этим вы увидите траекторию движения заряда. Если произошло попадание, корпус тарелки озаряется вспышкой. Информация о количестве выстрелов и попаданий выводится на табло в левой части пульта управления звездолетом. После завершения сеанса программа предлагает начать игру заново.

Как и в случае с калькулятором, хочу предупредить: полный код симулятора в книге приведен не будет из-за его большого объема. Мы разберемся с разметкой страницы, объясним логику работы функций, а весь текст программы можно посмотреть, щелкнув на ссылке «Код примера», расположенной на демонстрационной странице.

Начнем с общих настроек страницы:

```
body {margin: 0px; font-family: Tahoma; font-size: 14px;}
```

Разместим первый элемент – основное изображение, служащее фоном игры:

```
<img id="osn" src="osn.jpg" alt="Кабина">
```

Его свойства:

```
#osn {position: absolute; top: 0px; left: 0px; z-index: 1;}
```

Роль системы прицеливания играет картинка **pric.gif**:

```
<img id="pricel" src="pric.gif" alt="Прицел">
```

```
#pricel {position: absolute; top: 340px; left: 340px; z-index: 4;}
```

Летающая тарелка:

```
<img id="nlo" src="nlo.png" alt="НЛО">
```

```
#nlo {position: absolute; top: 170px; z-index: 5; visibility: hidden;}
```

Мигающий сигнал предупреждения об атаке НЛО:

```
<img id="ataka" src="ataka.gif" alt="Атака">
```

```
#ataka {position: absolute; top: 385px; left: 580px; z-index: 2; visibility: hidden;}
```

Энергетический заряд, выпускаемый в момент выстрела:

```
<img id="torp" src="torp.jpg" alt="Торпеда">

#torp {position: absolute; top: 340px; width: 6px; height: 8px;
z-index: 3; visibility: hidden;}
```

Картинка, изображающая взрыв при попадании в НЛО:

```
<img id="bah" src="babah.gif" alt="Взрыв">

#bah {position: absolute; top: 110px; z-index: 5; visibility: hidden;}
```

Сообщение о завершении игры:

```
<b id="start">ИГРА ОКОНЧЕНА
<br><u>НАЧАТЬ ЗАНОВО</u></b>

#start {position: absolute; top: 210px; left: 260px; z-index: 6;
font-size: 28px; color: #F9EB08; visibility: hidden;}

u {color: #FFFFFF;}
```

Счетчик попаданий в летающую тарелку:

```
<b id="chet"></b>

#chet {position: absolute; top: 390px; left: 45px; z-index: 2;
font-size: 12px; color: #FFFFFF;}
```

Счетчик выстрелов:

```
<b id="pif">Выстрелы - 0</b>

#pif {position: absolute; top: 425px; left: 45px; z-index: 2;
color: #F9EB08; visibility: hidden;}
```

Итак, все необходимые компоненты на странице. Приступаем к программированию.

Как всегда, первым делом регистрируем обработчики событий:

```
window.addEventListener("load", nlo);
window.addEventListener("mousemove", pricel);
window.addEventListener("click", pli);
window.addEventListener("load", function()
{
```

```
document.getElementById("start").addEventListener("mouseover", start);
});
```

Сразу после загрузки страницы у нас запускается функция **nlo()**, которая задает траекторию движения инопланетного корабля. Функция **pricel()** управляет перемещениями прицельного механизма. Функция **pli()** запускает энергетический заряд и выводит некоторые сообщения на экран пульта управления. Функция **polet()** контролирует полет заряда и, в случае меткого попадания, выводит сообщения о повреждениях НЛО. Наконец функция **start()** необходима для повторного запуска игры.

После регистрации обработчиков мы создаем целую группу переменных, которые будут хранить различные промежуточные данные, необходимые для корректной работы программы:

```
var i;
var m;
var pri;
var vis=0;
var fin=0;
var p=340;
var stop=0;
var sto;
var wse;
var nach=0;
```


Сначала рассмотрим функцию, управляющую движением НЛО:

```
function nlo()
{
i=Math.floor(Math.random() * 600)+10;
m=Math.floor(Math.random() * 170)+110;
document.getElementById("nlo").style.left=i+"px";
document.getElementById("nlo").style.top=m+"px";
document.getElementById("bah").style.visibility="hidden";
document.getElementById("nlo").style.visibility="visible";
document.getElementById("ataka").style.visibility="visible";
document.getElementById("pif").style.visibility="visible";
sto=window.setTimeout(nlo, 1500);
}
```

Объект Math *предоставляет программисту доступ к свойствам и методам, позволяющим выполнять сложные математические вычисления. В частности, метод* ***Math.random()*** *генерирует случайное число от 0 до 1, а метод* ***Math.floor()*** *округляет число до ближайшего нижнего целого значения (Например* ***Math.floor(2.8)*** *даст результат 2). В нашем случае последовательное применение двух методов необходимо для генерации случайных координат.*

В первых двух выражениях создаются два случайных числа – они выполняют роль координат летающей тарелки в каждом цикле функции **nlo()**. Следующие два оператора присваивают эти координаты свойствам **left** и **top** изображения НЛО. Затем свойствам **visibility** нескольких элементов игры присваиваются необходимые значения – **hidden** или **visible**. Когда происходит попадание в летающую тарелку, отдельные слои становятся невидимыми, другие, наоборот, видимыми, поэтому для продолжения игры необходимо вернуть им первоначальное состояние. Последняя инструкция запускает функцию через каждые полторы секунды, поэтому игрок видит скачкообразное перемещение инопланетного звездолета.

Движением прицела (он перемещается только горизонтально) управляет функция **pricel()**:


```
function pricel(event)
{
var e=event;
var x=e.clientX;
if(687<x)
  {
  document.getElementById("pricel").style.left=662+"px";
  }
else if(x<60)
  {
  document.getElementById("pricel").style.left=35+"px";
  }
else
  {
  document.getElementById("pricel").style.left=event.x-26+"px";
  }
}
```


Функция выполняет простые действия: присваивает свойству **left** рисунка скорректированную горизонтальную координату мыши и останавливает перемещение прицела, если мышь сместилась за границы игрового поля. Числовые параметры функции выбраны такими, чтобы указатель мыши находился примерно по центру прицела, на одном уровне с мигающей точкой.

Когда игрок щелкает мышью, вызывается функция **pli()**:

```
function pli(event)
{
var e=event;
var x=e.clientX;
if(stop==0)
  {
```

```
  if(nach==0)
    {
    if(685<x)
      {
      pri=684;
      }
    else if(x<55)
      {
      pri=57;
      }
    else
      {
      pri=x-3;
      }
    vis++;
    if(vis==16)
      {
      window.clearTimeout(sto);
      window.clearTimeout(wse);
      document.getElementById("pif").style.left="30px";
      document.getElementById("chet").innerHTML="ВАС СБИЛИ !";
      document.getElementById("pif").innerHTML="ЭНЕРГИЯ
ИСЧЕРПАНА !";
      document.getElementById("ataka").style.visibility="hidden";
      document.getElementById("start").style.visibility="visible";
      stop=1;
      }
    else
      {
      document.getElementById("torp").style.left=pri+"px";
      document.getElementById("torp").style.visibility="visible";
      document.getElementById("pif").innerHTML="Выстрелы - "+vis;
      nach=1;
      polet();
      }
    }
  }
}
```

*Метод **clearTimeout()** отменяет запланированное ранее выполнение функции, которая запускается таймером **setTimeout()**. Можно также сказать, что метод останавливает работу таймера.*

Первые операторы функции вычисляют горизонтальную точку, из которой будет выпущен поражающий заряд. Затем следуют два блока инструкций.

Разберем сначала нижний. Первая инструкция нижнего блока устанавливает стартовое положение заряда, вторая делает видимым его изображение, третья выводит на табло количество выстрелов. После этого идентификатору **nach** присваивается значение 1 – за счет этого повторный выстрел будет невозможен до тех пор, пока уже выпущенный заряд не уйдет за пределы экрана или не произойдет попадание. Последняя инструкция запускает функцию **polet()**.

Теперь посмотрим на верхний блок инструкций. Он выполняется, когда у земного корабля исчерпан энергетический ресурс (то есть сделано больше 15 выстрелов). Первым делом останавливаются таймеры функций **nlo()** и **polet()** – в результате их выполнение прерывается. Затем выводятся предупреждения о поражении игрока и гаснет надпись «Атака НЛО». Появляется сообщение «ИГРА ОКОНЧЕНА» и ссылка «НАЧАТЬ ЗАНОВО». Переменной **stop** присваивается значение 1, благодаря чему налагается запрет на выстрелы после завершения игры.

Наиболее «объемной» получилась функция **polet()**:

```
function polet()
{
if(p>120)
  {
  if((p<m)&&(p>m-50)&&(pri<i+105)&&(pri>i))
    {
    window.clearTimeout(sto);
    window.clearTimeout(wse);
    document.getElementById("nlo").style.visibility="hidden";
    document.getElementById("torp").style.visibility="hidden";
    document.getElementById("bah").style.top=m+"px";
    document.getElementById("bah").style.left=i+"px";
    document.getElementById("bah").style.visibility="visible";
    nach=0;
    fin++;
    if(fin==1)
      {
      document.getElementById("chet").innerHTML="Повреждения НЛО -
20%";
      sto=window.setTimeout(nlo, 1200);
      p=340;
      }
    if(fin==2)
      {
      document.getElementById("chet").innerHTML="Повреждения НЛО -
40%";
      sto=window.setTimeout(nlo, 1200);
      p=340;
      }
```

```
    if(fin==3)
      {
      document.getElementById("chet").innerHTML="Повреждения НЛО -
60%";
      sto=window.setTimeout(nlo, 1200);
      p=340;
      }
    if(fin==4)
      {
      document.getElementById("chet").innerHTML="Повреждения НЛО -
80%";
      sto=window.setTimeout(nlo, 1200);
      p=340;
      }
    if(fin==5)
      {
      document.getElementById("chet").innerHTML="НЛО СБИТ !";
      document.getElementById("ataka").style.visibility="hidden";
      document.getElementById("start").style.visibility="visible";
      stop=1;
      }
    }
  else
    {
    p=p-10;
    document.getElementById("torp").style.top=p+"px";
    wse=window.setTimeout(polet, 20);
    }
  }
else
  {
  document.getElementById("torp").style.visibility="hidden";
  nach=0;
  p=340;
  window.clearTimeout(wse);
  }
}
```

До тех пор, пока заряд не достиг границ экрана (то есть верно условие **p>120**), полетом энергетической субстанции управляет блок

```
p=p-10;
document.getElementById("torp").style.top=p+"px";
wse=window.setTimeout(polet, 20);
```

Каждые 20 миллисекунд заряд смещается вверх на 10 пикселей. Если он так и не попал в летающую тарелку, то выполняется блок инструкций

```
document.getElementById("torp").style.visibility="hidden";
nach=0;
p=340;
window.clearTimeout(wse);
```

Заряд становится невидимым, его изображение оказывается в стартовом вертикальном положении (**p=340**). Идентификатор **nach** получает значение 0 – тем самым разрешается следующий выстрел. Последняя инструкция останавливает выполнение функции **polet()**.

Если заряд попал в НЛО, то есть набор условий

```
if((p<m)&&(p>m-50)&&(pri<i+105)&&(pri>i))
```

оказался выполненным, то первым делом останавливаются таймеры функций **nlo()** и **polet()**. Летающая тарелка и заряд исчезают с экрана, а в точке попадания появляется изображение взрыва:

```
document.getElementById("bah").style.top=m+"px";
document.getElementById("bah").style.left=i+"px";
document.getElementById("bah").style.visibility="visible";
```

Дальше следуют инструкции, которые выводят на табло земного корабля сообщения об уровне повреждений НЛО, Если этот уровень не превышает 80%, то игра продолжается. Если произошло 5 попаданий, выполняется блок

```
document.getElementById("chet").innerHTML="НЛО СБИТ !";
document.getElementById("ataka").style.visibility="hidden";
document.getElementById("start").style.visibility="visible";
stop=1;
```

Появляется информация «НЛО СБИТ», предупреждение об атаке инопланетного корабля гаснет. Одновременно выводится сообщение «ИГРА ОКОНЧЕНА» и ссылка «НАЧАТЬ ЗАНОВО». Переменной **stop** присваивается значение 1, что приводит к запрету выстрелов после завершения игры.

У нас осталась последняя функция **start()**, которая запускается, когда вы наводите указатель мыши на ссылку «НАЧАТЬ ЗАНОВО»:

```
function start()
{
document.getElementById("start").style.visibility="hidden";
document.getElementById("ataka").style.visibility="visible";
document.getElementById("chet").innerHTML="";
```

```
document.getElementById("pif").style.left="45px";
document.getElementById("pif").innerHTML="Выстрелы - 0";
vis=0;
fin=0;
p=340;
stop=0;
nlo();
}
```

Задача функции – установить переменным начальные значения, обнулить счетчики на табло земного звездолета, скрыть сообщение об окончании игры и показать предупреждение «Атака НЛО». Последняя инструкция запускает игру заново.

Хотя на первый взгляд программа содержит множество инструкций, операторов и выражений, принцип ее действия оказался не очень сложным. Изучая фрагменты листинга и логику действия функций, вы получили представление о создании простейшей игры на JavaScript. Надеюсь, это придаст вам энтузиазма и смелости, необходимых для создания собственных игр.

Заключение

Вот и все, о чем я хотел рассказать на страницах этой книги. Наше повествование – у финишной черты. Мы рассмотрели множество случаев применения JavaScript. Но в жизни таких случаев несравненно больше. Автор скромно надеется, что теперь его читатели ощущают себя вооруженными ценным опытом, полученным во время изучения, копирования и совершенствования примеров из книги. А значит, они смелее и увереннее будут браться за решение других, часто весьма непростых задач.

Мне со своей стороны остается пожелать вам оригинальных программ и безошибочного кода. Удачи!

Валерий Викторович ЯНЦЕВ

JAVASCRIPT

ГОТОВЫЕ ПРОГРАММЫ

Учебное пособие

Зав. редакцией
литературы по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *Н. А. Кривилёва*
Корректор *Н. Н. Бутарова*
Выпускающий *В. А. Иутин*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com
196105, Санкт-Петербург, пр. Юрия Гагарина, д.1, лит. А.
Тел.: (812) 336-25-09, 412-92-72.
Бесплатный звонок по России: 8-800-700-40-71

ГДЕ КУПИТЬ

ДЛЯ ОРГАНИЗАЦИЙ:

Для того, чтобы заказать необходимые Вам книги, достаточно обратиться в любую из торговых компаний Издательского Дома «ЛАНЬ»:

по России и зарубежью
«ЛАНЬ-ТРЕЙД». 196105, Санкт-Петербург, пр. Юрия Гагарина, 1, лит. А
тел.: (812) 412-85-78, 412-14-45, 412-85-82; тел./факс: (812) 412-54-93
e-mail: trade@lanbook.ru; ICQ: 446-869-967
www.lanbook.com
пункт меню **«Где купить»**
раздел **«Прайс-листы, каталоги»**

в Москве и в Московской области
«ЛАНЬ-ПРЕСС». 109387, Москва, ул. Летняя, д. 6
тел.: (499) 722-72-30, (495) 647-40-77; e-mail: lanpress@lanbook.ru

в Краснодаре и в Краснодарском крае
«ЛАНЬ-ЮГ». 350901, Краснодар, ул. Жлобы, д. 1/1
тел.: (861) 274-10-35; e-mail: lankrd98@mail.ru

ДЛЯ РОЗНИЧНЫХ ПОКУПАТЕЛЕЙ:

интернет-магазин
Издательство «Лань»: http://www.lanbook.com

магазин электронных книг
Global F5: http://globalf5.com/

Подписано в печать 02.02.21.
Бумага офсетная. Гарнитура Школьная. Формат 70×100 $^{1}/_{16}$.
Печать офсетная. Усл. п. л. 16,25. Тираж 30 экз.

Заказ № 165-21.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские технологии»
109316, г. Москва, Волгоградский пр., д. 42, к. 5.