Создание слайдера, подсказок, форм и других динамических элементов для сайта

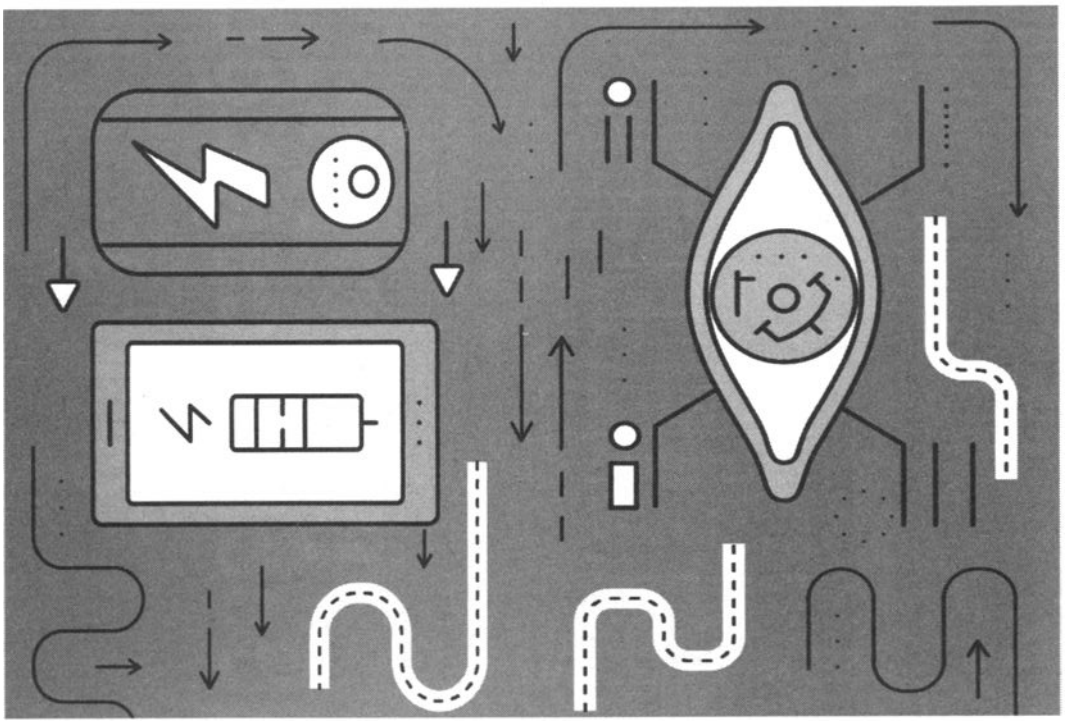
Стек полезных инструментов и технологий: AJAX, JQuery

JavaScript НА ПРИМЕРАХ

ПРАКТИКА, ПРАКТИКА И ТОЛЬКО ПРАКТИКА

"Наука и Техника"

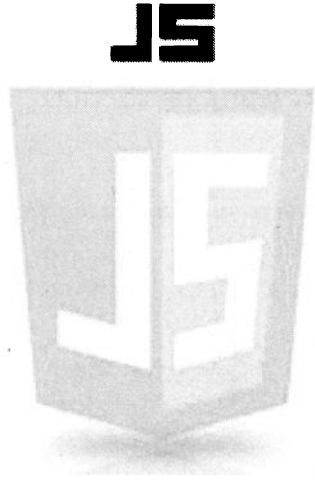
Санкт-Петербург



Никольский А. П.

JavaScript на примерах

Практика, практика и только практика



"Наука и Техника"

Санкт-Петербург

УДК 004.738

ББК 32.973

ISBN 978-5-94387-762-9

Никольский А. П.

JAVASCRIPT НА ПРИМЕРАХ. ПРАКТИКА, ПРАКТИКА И ТОЛЬКО ПРАКТИКА — СПб.: Наука и Техника, 2018. — 272 с., ил.

Серия «На примерах»

Эта книга является превосходным учебным пособием для изучения языка программирования JavaScript на примерах. Изложение ведется последовательно: от написания первой программы, до создания полноценных проектов: интерактивных элементов (типа слайдера, диалоговых окон) интернет-магазина, лендинговой страницы и проч. По ходу даются все необходимые пояснения и комментарии.

Книга написана простым и доступным языком. Лучший выбор для результативного изучения JavaScript!

Материалы к книге размещены на сайте издательства "Наука и Техника" в разделе "Материалы к книгам".

ISBN 978-5-94387-762-9

Контактные телефоны издательства:
(812) 412 70 26

Официальный сайт: www.nit.com.ru

Содержание

Введение

Давно уже прошло время статических веб-страниц. Ведь на дворе век автоматизации и любые, даже самые простые, действия принято автоматизировать. В случае с веб-страницами автоматизация достигается или на стороне сервера или на стороне клиента. Также возможна автоматизация, как на стороне сервера, так и на стороне клиента.

Давайте разберемся, что к чему. Представим, у нас есть некоторая база данных, скажем список товаров интернет-магазина, и нам нужно ее вывести на нашем сайте. Теоретически, список товаров можно оформить в виде статической HTML-страницы. Но обновлять такой магазин будет очень неудобно - ведь со временем нужно будет добавить/удалить товары, изменить цены и т.д. Гораздо проще написать, скажем, на PHP сценарий, который будет подключаться к БД, выбирать товары из нужной категории (скажем, компьютеры, мобильные телефоны, бытовая техника и т.д.) и отображать их посетителю. Также с помощью PHP-сценария можно легко изменить все позиции сразу, например, в канун Нового года устроить небольшую акцию и снизить цены на 10%. Вручную отредактировать несколько тысяч записей, согласитесь, не просто. А простейший сценарий сделает это за доли секунды.

Это и есть автоматизация на стороне сервера, поскольку PHP-сценарий выполняется интерпретатором на сервере, а пользователь лишь видит результат выполнения этого сценария. PHP в этой книге не рассматривается, а если вы заинтересовались, на виртуальных полках книжных магазинах вы найдете множество книг, посвященных этому языку программирования.

Теперь переходим к автоматизации на стороне клиента. Итак, серверный сценарий сгенерировал список товаров. Пользователь хочет купить какой-то товар. Принцип работы большинства магазинов прост: посетитель добавляет все необходимые ему товары в корзину, а потом оформляет заказ или вообще отказывается от покупки. Вопрос заключается в том, как будет реализована корзина. Ее тоже можно реализовать как на стороне сервера, так и на стороне клиента. Скажем, для кнопки "Купить" сделать ссылку вида http://our_shop.com/buy.php?id=12345, где 12345 - это идентификатор (артикул) товара, который будет помещен в корзину. Сама же корзина пользователя будет храниться или в БД (если нужно отслеживать все заказы пользователя), или же во временном файле сессии в каталоге /tmp (этот файл будет удален, как только пользователь закроет браузер).

Но такой вариант работы с корзиной малоэффективен. Представим, что в среднем покупатель может заказать 2-3 товара (один основной и 1-2 аксессуара к нему, например ноутбук, а к нему — мышку и дополнительную акустику). А что если пользователей 1000 (для средних интернет-магазинов это не показатель)? Выходит, к серверу будет почти одновременно отправлено 2-3 тысячи запросов. Сервер должен их обработать и вывести результат (по сути ту же страницу, что и была, но со строкой "Товаров в корзине: число"). То есть ради изменения одного символа мы так нагружаем сервер процессора. Да и посетитель не будет рад, так как на перезагрузку страницы нужно время, а если сервер будет сильно загружен или у клиента низкоскоростной доступ к Интернету, ждать придется несколько секунд.

Оказывается, можно все автоматизировать нашу задачу на стороне клиента. Вместе с HTML-кодом списка товаров наш PHP-сценарий отправит код JavaScript-сценария, который будет выполнен на клиенте, а не на сервере. Когда пользователь нажмет кнопку "Купить", идентификатор товара запишется в Cookies браузера (чтобы содержимое корзины не изменялось при переходе от одной до другой страницы магазина). При этом JavaScript-сценарий будет выполнен на компьютере пользователя. Страница не будет перезагружена, и все будут рады. И хостинг-провайдер - поскольку вы не нагружаете сервер лишними запросами, и пользователь, которому не придется ждать несколько секунд, пока перезагрузится страница.

Когда же пользователь захочет оформить заказ и нажмет соответствующую кнопку/ссылку, содержимое корзины будет отправлено на сервер для обработки заказа. Получается, что вместо 2-3 запросов среднестатистический пользователь сделает всего один (если не считать запросов GET, когда он будет просматривать содержимое интернет-магазина, но от этого никуда не денешься) - когда будет оформлять заказ. И вместо 2-3 тысяч запросов мы получим тысячу. Вот вам и оптимизация. Теоретически, можно уменьшить и число GET-запросов (когда пользователь просматривает категории магазина, а сценарий на сервере генерирует содержимое той или иной категории в формате HTML), но не думаю, что об этом стоит говорить во введении.

Сценарий на стороне сервера, как уже отмечено, вместе с HTML-страницей может передать и JavaScript-код, который лежит в основе автоматизации на стороне клиента. Справедливости ради нужно отметить, что кроме JavaScript допускается использование и других скриптинговых языков, например VBScript. Однако в этой главе мы будем рассматривать только JavaScript, который наиболее популярен среди веб-мастеров.

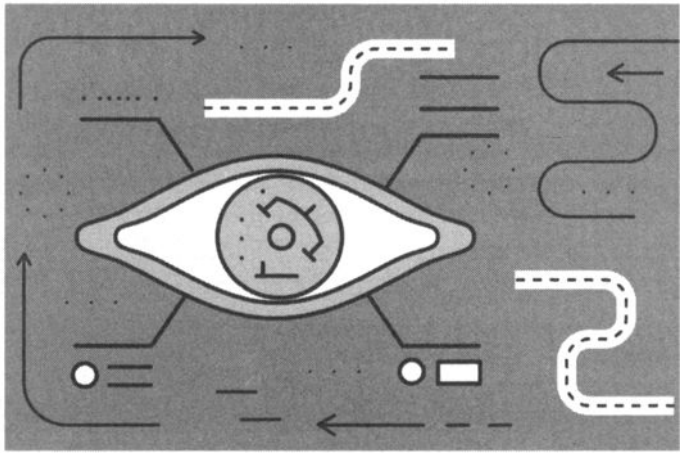
Код JavaScript находится в составе HTML-документа (или выносится в отдельный файл, если код слишком большой). Сам же JavaScript также может

генерировать HTML-код (кроме выполнения вычислительных операций и отображения всевозможных запросов и сообщений).

HTML-код в этой книге мы не рассматриваем, считается, что читатель с ним знаком. Данная книга - самое что ни есть практическое руководство по JavaScript, позволяющее выучить этот скриптовый язык с нуля. Начнем мы с создания обычного HTML-сайта, а затем попытаемся улучшить его средствами JavaScript. Теоретические и практические главы в этой книге будут перемешаны - чтобы читатель не уставал от теории и мог сразу опробовать полученные знания на практике.

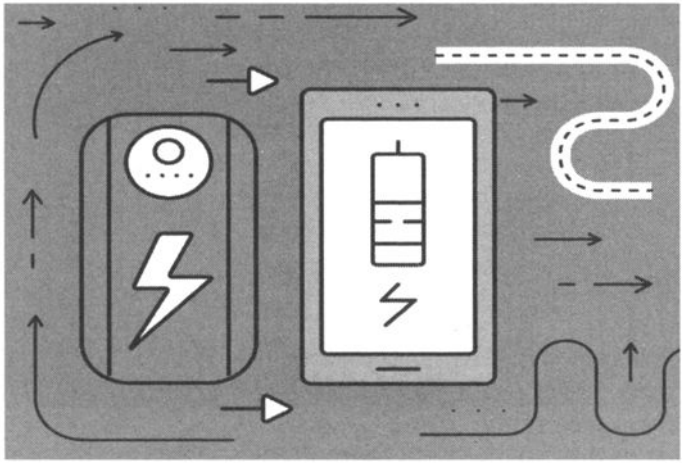
Что же касается написания кода с 0, то в книге мы будем как писать полностью собственный код, так и использовать уже готовые библиотеки (когда это целесообразно). Например, вы хотите сделать обработку формы, чтобы не подключать тяжеловесные библиотеки, можно написать код вручную - тем более, что там ничего сложного нет. Совсем другое дело, когда вам нужно реализовать более объемную функцию, например галерею. Нет смысла тратить недели времени на разработку галереи, биться над одинаковым отображением кода и стилей в разных браузерах, если можно использовать уже готовое решение. Зачем изобретать велосипед заново?

Приятного чтения!



Глава 1.

Простой сайт без JavaScript



Цель первой главы - разработать простенький сайт, который мы сможем потом усовершенствовать средствами JavaScript. Сайт будет написан на HTML с применением каскадных таблиц стилей (CSS) для придания ему относительно привлекательного вида. Наш сайт будет посвящен набирающим в данное время популярность устройствам - гироскутерам.

1.1. Главная страница

На главной странице у нас будет размещена информационная область (ее модно называть featured) и некоторые товары (например, самые популярные или те, которые нам нужно продать в первую очередь).

Как правило, в информационной области можно разместить объявления о всевозможных акциях, добавить в нее слайдер с лучшими продуктами или новинками и т.д. Мы просто дополним краткую информацию о преимуществах гироскутеров (лист. 1.1).

Листинг 1.1. Преимущества гироскутеров

```
<div id="body">
     <div id="featured">
        <img src="images/main.jpg" alt="">
        <div>
           <h2>Преимущества</h2>
           <p><span>Экологичность</span>
           <p><span>Экономичность</span>
           <p><span>Компактность</span>
           <p><a href="dostavka.html" class="more">купить</a>
        </div>
     </div>
```

Далее последует информация о товарах (лист. 1.2). На главной мало кто публикует весь товар (если самого товара предостаточно), поэтому мы ограничимся тремя единицами.

Листинг 1.2. Некоторые товары

```
<ul>
   <li>
      <a href="catalog.html">
         <img src="images/9.jpg" alt="">
         <br>IO CHIC Smart-LS 9" Black
      </a>
   </li>
   <li>
      <a href="catalog.html">
         <img src="images/20.jpg" alt="">
         <br>IO CHIC Cross 20" Black
      </a>
   </li>
   <li>
      <a href="catalog.html">
         <img src="images/6.jpg" alt="">
         <br>IO CHIC Fairy 6.7" Red
      </a>
   </li>
</ul>
```

Понятное дело, у нашего сайта будут меню и "подвал" - нижняя часть сайта. Код меню и "подвала" приводится в листинге 1.3. Этот же листинг отображает весь код главной страницы - index.html.

Листинг 1.3. Код index.html

```
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>Гироскутеры</title>
   <link rel="stylesheet" type="text/css" href="css/style.css">

</head>
<body>
   <div id="header">
      <h1>Гироскутеры на любой вкус</h1>
      <ul id="navigation">
         <li class="selected">
            <a href="index.html">главная</a>
         </li>
         <li>
```

```
        <a href="about.html">доставка и оплата</a>
      </li>
      <li>
        <a href="gallery.html">каталог</a>
      </li>

      <li>
        <a href="contact.html">контакты</a>
      </li>
    </ul>
  </div>
  <div id="body">
    <div id="featured">
      <img src="images/main.jpg" alt="">
      <div>
        <h2>Преимущества</h2>
        <p><span>Экологичность</span>
        <p><span>Экономичность</span>
        <p><span>Компактность</span>
        <p><a href="dostavka.html" class="more">купить</a>
      </div>
    </div>
    <ul>
      <li>
        <a href="9.html">
          <img src="images/9.jpg" alt="">
          <br>IO CHIC Smart-LS 9" Black
        </a>
      </li>
      <li>
        <a href="20.html">
          <img src="images/20.jpg" alt="">
          <br>IO CHIC Cross 20" Black
        </a>
      </li>
      <li>
        <a href="6.html">
          <img src="images/6.jpg" alt="">
          <br>IO CHIC Fairy 6.7" Red
        </a>
      </li>
    </ul>
  </div>
  <div id="footer">
    <div>
      <p>&copy; 2016 Магазин гироскутеров.</p>
```

```
        <ul>
          <li>
            <a href="" id="twitter">twitter</a>
          </li>
          <li>
            <a href="" id="facebook">facebook</a>
          </li>
          <li>
            <a href="" id="googleplus">googleplus</a>
          </li>
          <li>
            <a href="" id="pinterest">pinterest</a>
          </li>
        </ul>
      </div>
    </div>
</body>
</html>
```

Как видите, в меню есть ссылки на другие страницы - "Доставка и оплата", "Каталог и Контакты". А в "подвале" сайта есть ссылки на наши страницы в социальных сетях.

1.2. Страница с информацией о товаре

Мы не будем приводить код всех страниц сайта, он сходен. Например, на страницах "Доставка и оплата" и "Контакты" выводится только текстовая информация со способами оплаты и контактной информацией. В каталоге отображаются изображения и названия продуктов, как мы это делали на главной странице. Щелчок по тому или иному товару открывает страницу с информацией о нем. Обычно такая страница содержит как минимум одно изображение товара, описание и характеристики. Код страницы товара приведен в листинге 1.4.

Листинг 1.4.

```
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>Гироскутеры</title>
   <link rel="stylesheet" type="text/css" href="css/style.css">
```

```
</head>
<body>
   <div id="header">
      <h1>Гироскутеры на любой вкус</h1>
      <ul id="navigation">
         <li class="selected">
            <a href="index.html">главная</a>
         </li>
         <li>
            <a href="about.html">доставка и оплата</a>
         </li>
         <li>
            <a href="gallery.html">каталог</a>
         </li>

         <li>
            <a href="contact.html">контакты</a>
         </li>
      </ul>
   </div>
   <div id="body">
      <h2>IO CHIC Smart-LS 9" Black</h2>
         <img src="images/9.jpg" alt="">

               <p align="left"><b>Описание</b>
               <p align="justify">Гироскутер CHIC Cross широко
используется в прокате для развлечения, в аэропортах,
международных конференц-центрах, выставочных залах, стадионах,
больших тематических парках и площадях, полях для гольфа, в
экотуризме, на многочисленных курортах и частных виллах, для
патрулирования коттеджных поселков, парков, питомников, в
развлекательных и рекламных мероприятиях.

<p align=justify>Гироскутер CHIC Cross имеет два режима
скорости: начинающий режим – максимум 8 км/ч, и обычный - 15-
20 км/ч. CHIC Cross имеет ключ, который дает доступ к кнопке
включения гироскутера, и брелок для оперативного управления
CHIC Cross.
    <p align="left"><b>Характеристики</b>


    <br>Максимальная рекомендуемая нагрузка: 125 кг
    <br>Мощность: 2000 Вт (2 x 1000 Вт)
    <br>Максимальная скорость: 20 км/ч
```

```
    <p align="left"><b>Цена</b>
     <br><b>4999$</b>
     <ul>
        <li>
           <a href="catalog.html">
              <img src="images/9.jpg" alt="">
              <br>IO CHIC Smart-LS 9" Black
           </a>
        </li>
        <li>
           <a href="catalog.html">
              <img src="images/20.jpg" alt="">
              <br>IO CHIC Cross 20" Black
           </a>
        </li>
        <li>
           <a href="catalog.html">
              <img src="images/6.jpg" alt="">
              <br>IO CHIC Fairy 6.7" Red
           </a>
        </li>
     </ul>
  </div>
  <div id="footer">
     <div>
        <p>&copy; 2016 Магазин гироскутеров.</p>
        <ul>
           <li>
              <a href="" id="twitter">twitter</a>
           </li>
           <li>
              <a href="" id="facebook">facebook</a>
           </li>
           <li>
              <a href="" id="googleplus">googleplus</a>
           </li>
           <li>
              <a href="" id="pinterest">pinterest</a>
           </li>
        </ul>
     </div>
  </div>
</body>
</html>
```

1.3. Стили

Если вы сейчас просмотрите обе созданные нами страницы, то выглядеть они будут очень непривлекательно. Чтобы они хоть как-то смотрелись, нужно задать оформление в CSS-файле style.css, который подключен к обеим страницам (лист. 1.5).

Листинг 1.5. Файл style.css

```
@font-face {
   font-family: 'poller_oneregular';
   src: url('../fonts/pollerone/pollerone-webfont.eot');
   src: url('../fonts/pollerone/pollerone-webfont.eot?#iefix')
format('embedded-opentype'),
       url('../fonts/pollerone/pollerone-webfont.woff') format('woff'),
       url('../fonts/pollerone/pollerone-webfont.ttf')
format('truetype'),
       url('../fonts/pollerone/pollerone-webfont.svg#poller_oneregular')
format('svg');
}
@font-face {
   font-family: 'leckerli_oneregular';
   src: url('../fonts/leckerlione/leckerlione-regular-webfont.eot');
   src: url('../fonts/leckerlione/leckerlione-regular-webfont.
eot?#iefix') format('embedded-opentype'),
       url('../fonts/leckerlione/leckerlione-regular-webfont.woff')
format('woff'),
       url('../fonts/leckerlione/leckerlione-regular-webfont.ttf')
format('truetype'),
       url('../fonts/leckerlione/leckerlione-regular-webfont.
svg#leckerli_oneregular') format('svg');
}

body {
   background: #fff;
   font-family: Arial, Helvetica, sans-serif;
   font-size: 14px;
   font-weight: normal;
   line-height: 1;
   margin: 0;
   min-width: 960px;
   padding: 0;
}
p {
   color: #252525;
   line-height: 24px;
```

```
    margin: 0;
    padding: 0;
}
p a {
    color: #252525;
    text-decoration: underline;
}
p a:hover {
    color: #898989;
}
#header {
    margin: 0 auto;
    padding: 44px 0 50px;
    text-align: center;
    width: 960px;
}
#header a.logo {
    display: block;
    margin: 0 auto;
    padding: 0;
    width: 340px;
}
#header a.logo img {
    border: 0;
    display: block;
    margin: 0;
    padding: 0;
}
#header ul {
    margin: 0;
    padding: 56px 0 0;
}
#header ul li {
    display: inline;
    list-style: none;
    margin: 0;
    padding: 0 19px;
}
#header ul li a {
    color: #252525;
    display: inline-block;
    font-family: "Arial Black", Gadget, sans-serif;
    font-size: 14px;
    font-weight: normal;
    margin: 0;
    padding: 0 0 3px;
```

```
    text-decoration: none;
    text-transform: uppercase;
}
#header ul li a:hover, #header ul li.selected a {
    color: #0ba39c;
}
#body {
    margin: 0;
    padding: 0 0 30px;
    text-align: center;
}
#body h1 {
    color: #fff;
    display: inline;
    font-family: poller_oneregular;
    font-size: 35px;
    font-weight: normal;
    margin: 0;
    padding: 0;
    text-transform: uppercase;
}
#body h1 span {
    background: #252525;
    display: inline-block;
    padding: 15px 28px 12px;
}
#body > div {
    margin: 0 auto;
    padding: 0;
    width: 960px;
}
#body > div img {
    border: 2px solid #292929;
    margin: 24px 0 0;
    padding: 0;
}
#body > div .article {
    margin: 0 auto;
    padding: 30px 30px 0;
    width: 560px;
}
#body > div .article h2 {
    font-family: "Arial Black", Gadget, sans-serif;
    font-size: 25px;
    font-weight: normal;
    margin: 0 0 30px;
```

```
}
#body > div .article h3 {
   color: #252525;
   line-height: 24px;
   margin: 0;
   padding: 0;
}
#body > div .article h4 {
   background: #252525;
   color: #fff;
   display: inline-block;
   font-family: "Arial Black", Gadget, sans-serif;
   font-size: 25px;
   font-weight: normal;
   margin: 0 0 30px;
   padding: 8px 27px 10px;
}
#body > div .article p {
   margin: 0 0 30px;
}
#body div ul {
   margin: 0;
   padding: 24px 10px 0;
   width: 940px;
}
#body div ul li {
   border: 2px solid #1a1a1a;
   display: block;
   list-style: none;
   margin: 0 0 13px;
   overflow: hidden;
   padding: 0;
}
#body div ul li .figure {
   border-right: 2px solid #1a1a1a;
   float: left;
   margin: 0;
   padding: 0;
   width: 376px;
}
#body div ul li img {
   border: 0;
   display: block;
   margin: 0;
   padding: 0;
   opacity: 0.8;
```

```
    transition: 0.5s ease-in-out;
}
#body div ul li .figure:hover img {
    opacity: 1;
}
#body div ul li div {
    float: left;
    margin: 0;
    padding: 63px 0 20px;
    width: 558px;
}
#body div ul li div h3 {
    color: #252525;
    font-family: "Arial Black", Gadget, sans-serif;
    font-size: 25px;
    font-weight: normal;
    margin: 0;
    padding: 0;
    text-transform: uppercase;
}
#body div ul li div p {
    padding: 30px 30px 18px;
}
#body div ul li div a.more {
    background: #000;
    color: #fff;
    display: inline-block;
    font-family: "Arial Black", Gadget, sans-serif;
    font-size: 18px;
    font-weight: normal;
    margin: 0;
    padding: 8px 28px;
    text-decoration: none;
    text-transform: uppercase;
}
#body div ul li div a.more:hover {
    color: #252525;
    background: #fff;
    border: 2px solid #252525;
}
#body #featured {
    margin: 0 0 13px;
    padding: 0;
    position: relative;
    width: 100%;
}
```

```
#body #featured img {
   border: 0;
   display: block;
   margin: 0;
   padding: 0;
   width: 100%;
}
#body #featured div {
   left: 50%;
   margin-left: -420px;
   margin-top: -171px;
   padding: 0 0 0 470px;
   position: absolute;
   top: 50%;
   text-align: left;
   width: 490px;
}
#body #featured div h2 {
   background: #abf;
   color: #000;
   display: inline-block;
   font-family: poller_oneregular;
   font-size: 20px;
   font-weight: normal;
   margin: 0 0 18px;
   padding: 23px 27px 20px;
   text-transform: uppercase;
}
#body #featured div span {
   background: #252525;
   color: #fff;
   display: inline-block;
   font-size: 20px;
   margin: 7px 0;
   padding: 12px 27px;
}
#body #featured div a {
   background: #0ba39c;
   color: #fff;
   display: inline-block;
   font-size: 16px;
   margin: 15px 0 0;
   padding: 26px 27px 22px;
   text-align: center;
   text-decoration: none;
   text-transform: uppercase;
```

```
}
#body #featured div a:hover {
   background: #1fc3bc;
}
#body > ul {
   margin: 0 auto;
   overflow: hidden;
   padding: 24px 0 0;
   width: 960px;
}
#body > ul li {
   display: block;
   float: left;
   margin: 0 10px 20px;
   padding: 0;
   width: 300px;
}
#body > ul li a {
   color: #000;
   display: block;
   font-family: poller_oneregular;
   font-size: 20px;
   font-weight: normal;
   margin: 0;
   padding: 0;
   text-align: left;
   text-decoration: none;
   text-transform: uppercase;
}
#body > ul li a img {
   border: 2px solid #1a1a1a;
   display: block;
   margin: 20;
   padding: 0;
   opacity: .8;
   transition: 0.5s ease-in-out;
}
#body > ul li a:hover img {
   opacity: 1;
}
#body > ul li a span {
   display: block;
   margin: 0;
   padding: 33px 0 0;
}
#body form {
```

```
  margin: 0 auto;
  padding: 24px 0 0;
  width: 620px;
}
#body form input, #body form textarea {
  border: 2px solid #252525;
  color: #252525;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 15px;
  font-weight: normal;
  margin: 0 0 37px;
  padding: 20px 10px;
  text-align: center;
  text-transform: uppercase;
  width: 596px;
}
#body form textarea {
  height: 175px;
  overflow: auto;
  resize: none;
}
#body form #send {
  background: #252525;
  color: #fff;
  font-family: "Arial Black", Gadget, sans-serif;
  font-size: 15px;
  font-weight: normal;
  display: inline-block;
  padding: 20px 28px;
  width: auto;
}
#body form #send:hover {
  background: #fff;
  color: #252525;
  cursor: pointer;
}
#footer {
  background: #252525;
  margin: 0;
  padding: 36px 0 42px;
}
#footer div {
  margin: 0 auto;
  overflow: hidden;
  padding: 0;
  width: 960px;
```

```
}
#footer div p {
   color: #fff;
   float: left;
   line-height: 44px;
   margin: 0 0 0 10px;
   padding: 0;
}
#footer div ul {
   display: block;
   float: right;
   margin: 0;
   overflow: hidden;
   padding: 0;
   width: 256px;
}
#footer div ul li {
   display: block;
   float: left;
   list-style: none;
   margin: 0 10px;
   padding: 0;
   width: 44px;
}
#footer div ul li a {
   background: url(../images/icons.jpg) no-repeat 0 0;
   display: block;
   height: 44px;
   margin: 0;
   padding: 0;
   text-indent: -99999px;
   width: 44px;
}
#footer div ul li #twitter {
   background-position: 0 0;
}
#footer div ul li #twitter:hover {
   background-position: -44px 0;
}
#footer div ul li #facebook {
   background-position: 0 -44px;
}
#footer div ul li #facebook:hover {
   background-position: -44px -44px;
}
#footer div ul li #googleplus {
```

```
  background-position: 0 -88px;
}
#footer div ul li #googleplus:hover {
  background-position: -44px -88px;
}
#footer div ul li #pinterest {
  background-position: 0 -132px;
}
#footer div ul li #pinterest:hover {
  background-position: -44px -132px;
}
```

Вот теперь, подключив стили, можно взглянуть, что у нас получилось. На рис. 1.1 показано, как выглядит главная страница, на рис. 1.2 показана нижняя часть главной страницы - некоторые товары, а также "подвал" с кнопками социальных сетей.

Рис. 1.1. Главная страница

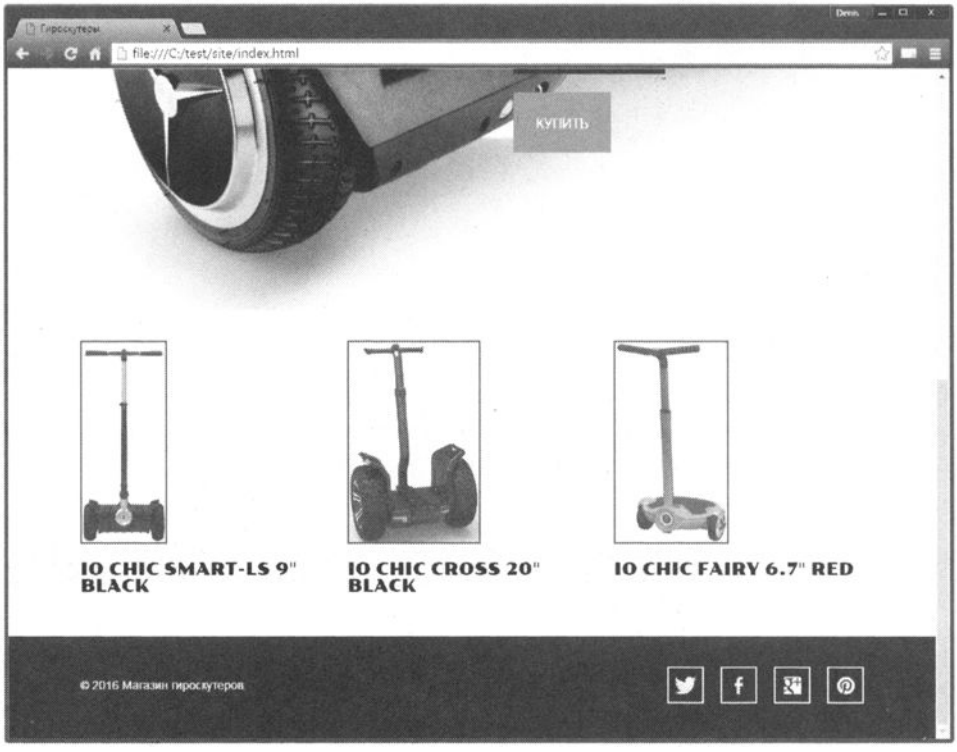


Рис. 1.2. Нижняя часть главной страницы

На рис. 1.3 показана страница товара - есть изображения товара, его описание, характеристики и цена. Внизу страницы приводятся ссылки на другие товары.

1.4. Недостатки нашего решения

У нашего решения очень много недостатков. Во-первых, посмотрите на меню. Нет, его основной недостаток даже не в том, что оно не динамическое, как мы привыкли видеть на современных сайтах. Его нужно включать в каждую страницу нашего сайта (как и подвал), поскольку HTML не содержит средств включения (include) других HTML-файлов. Когда страниц немного, это, в общем, не проблема. А представьте, что у вашего сайта есть уже несколько сотен страниц и вам нужно изменить меню или "подвал"? Тогда вам придется изменять все эти несколько сотен страниц. Это ужасно!

Во-вторых, нет никакой возможности придать "жизни" нашему сайту. Например, на главной странице не выводить постоянно одни и те же три то-

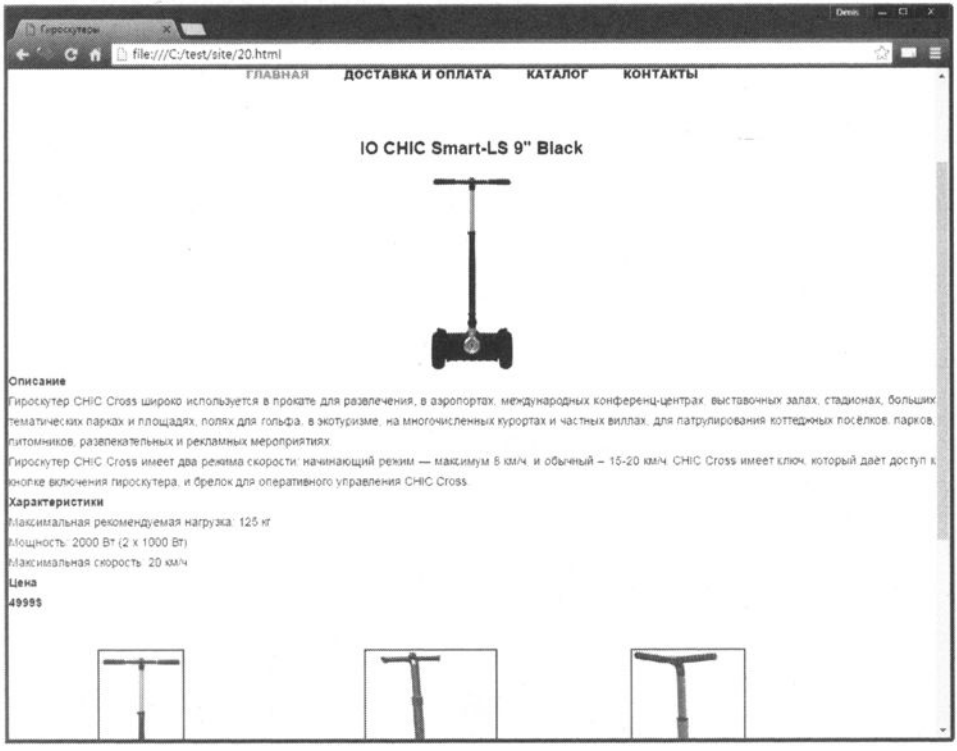


Рис. 1.3. Страница товара

вара, а вывести разные товары или разные изображения одних и тех же товаров. Нет возможности организовать слайдер товаров. Наша главная страница статична, и смотрится это ужасно, особенно, учитывая, что наш магазин продает динамические гироскутеры. Ладно бы, если мы продавали классические деревянные стулья...

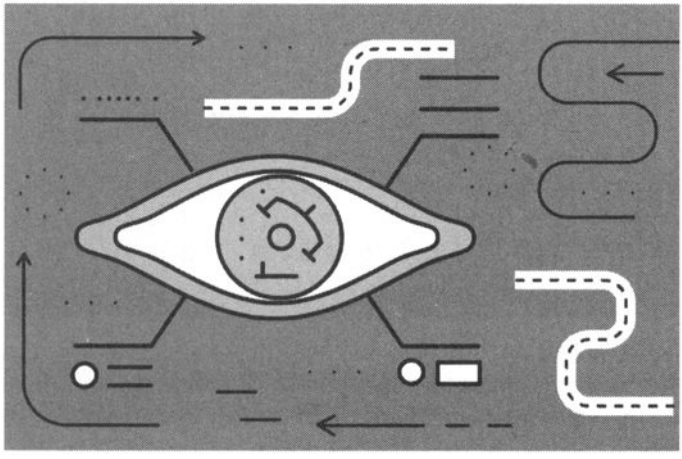
В-третьих, посмотрите на страницу с информацией о продукте. Галереи изображений нет. Организовать ее средствами HTML нельзя. Единственно возможное - это скачать из Интернета еще несколько изображений товара и добавить их рядом с имеющимся. Но это вы можете сделать и сами.

Описание, характеристики и цена товара указаны непривлекательно. Лично я вижу их в виде вкладок с информацией.

Кнопку "Купить" в нашем случае я даже и не делал. Особого смысла в этом нет, поскольку все, что можно сделать, - это ссылку на страницу "Доставка и оплата", как это было сделано на главной странице.

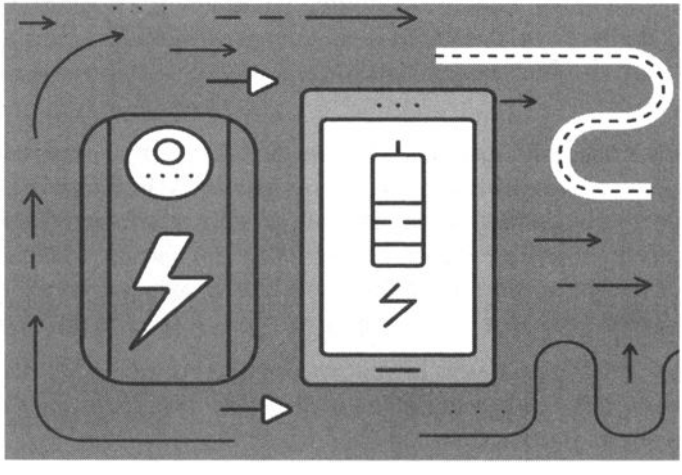
В последующих главах мы исправим все недостатки нашего сайта и превратим его в простейший интернет-магазин по продаже гироскутеров.


JavaScript™



Глава 2.

Основные понятия и первая программа



2.1. JavaScript - не Java

Прежде чем вы приступите к изучению JavaScript, нужно понимать, что Java – это не Java. Это два совершенно разных языка программирования. JavaScript не имеет ничего общего с языком Java, кроме похожего синтаксиса. Однако JavaScript может обращаться к внешним свойствам и методам Java-апплетов, которые встроены в HTML-страницу.

Java - это объектно-ориентированный язык программирования, а Java-апплеты запускаются с помощью компилятора. Апплеты Java встраиваются в Web-страницу, но хранятся на диске как отдельные файлы. Это двоичные файлы, и если вы их откроете, то не увидите исходный код апплета.

Сценарии JavaScript размещаются внутри Web-страницы и не могут существовать отдельно от нее. Для выполнения JS-сценариев не нужен компилятор, они выполняются браузером. JS-сценарий - это обычный текст, и вы можете просмотреть код сценария невооруженным взглядом – без какого-нибудь специального программного обеспечения.

Что же такое JavaScript? Это язык программирования с синтаксисом, сходным с языком Java, использующийся в составе HTML-страниц для увеличения их функциональности. Первоначально язык JavaScript был разработан компаниями Netscape и Sun Microsystems, за его основу был взят язык Sun Java.

JavaScript позволяет реализовать те функции страницы, которые невозможно реализовать стандартными тегами HTML. Сценарии запускаются в результате наступления какого-нибудь события, например пользователь нажал кнопку или изменился размер окна. JavaScript имеет доступ к свойствам документа и свойствам браузера. Например, на JavaScript вы можете легко изменять заголовок окна браузера или текст в строке состояния.

Для написания сценариев JavaScript не нужно никакое специальное программное обеспечение – достаточно простого текстового редактора. Так что вы можете использовать свой любимый HTML-редактор для написания JavaScript-кода.

2.2. Объектная модель JavaScript

В JavaScript используется объектная модель документа, в рамках которой каждый HTML-контейнер можно рассматривать как совокупность свойств, методов и событий, происходящих в браузере. По сути, это связь между HTML-страницей и браузером. В этой главе мы не будем вникать в ее подробности, а желающие "бежать впереди паровоза" могут узнать больше по следующей ссылке: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model

Пока вам нужно знать, что есть старший класс Window, позволяющий обращаться к методам и свойствам HTML-страницы и браузера. Например, метод close() позволяет закрыть окно браузера, а свойство location - обратиться к адресной строке браузера.

Если вы не знакомы с объектно-ориентированным программированием, тогда, наверное, не особо понимаете, о чем идет речь, когда мы говорим об объектах и методах. Не расстраивайтесь, пока принимайте все как есть, а дальше вы поймете что к чему. В JavaScript все основано на классах и объектах (поскольку это объектно-ориентированный язык), и без них вы не сможете написать свои программы. Именно поэтому мы начали разговор об объектной модели JavaScript в этой вводной главе.

Объект можно воспринимать как совокупность данных и методов (функций) для их обработки. В JavaScript с некоторыми объектами также связываются определенные события.

Давайте разберемся, что такое объект, как говорится, "на пальцах". Представим, что объект - это человек. Путь наш объект называется Human. У такого объекта может быть масса характеристик - имя, пол, дата рождения и т.д. Все это называется свойствами объекта. Обратиться к свойству можно так: `объект.свойство`

Например:

```
Human.Sex = 'M';
Human.Name = 'Николай';
Numan.YearOfBirth = '1976';
```

Метод – это функция обработки данных. Например, метод, возвращаемый год рождения человека может называться GetYear. Обращение к методу производится так:

```
объект.метод(параметры);
```

Например:

```
Human.GetYear();
```

Связать метод с функцией можно так:

```
объект.метод = имя_функции;
```

С объектом может быть связано какое-нибудь событие. Например, при рождении человека может генерироваться событие OnBirth. Для каждого события можно определить его обработчик – функцию, которая будет на него реагировать. Что будет делать эта функция, зависит от события. Например, обработчик OnBirth может заносить в некую общую таблицу базы данных информацию об объекте - имя, пол и дату рождения. Такая таблица может использоваться для ускорения поиска нужного объекта.

Теперь рассмотрим еще одно, более абстрактное, понятие - класс. Класс - это образец объекта, если хотите - тип переменной объекта. Пусть мы разработали класс Human, тогда объект, то есть экземпляр класса Human, может называться Human1. Вы можете создавать несколько объектов класса Human – имена у них будут разными:

```
John = new Human;   // создаем объект John класса Human
Mary = new Human;   // создаем объект Mary класса Human
```

В этой книге мы еще поговорим о создании объектов, а сейчас приведенной информации вполне достаточно для восприятия следующего материала.

2.3. Первая программа

Чтобы ваша JavaScript-программа (или сценарий) запустилась, ее нужно внедрить в HTML-код документа (или "связать" программу с документом). Обратите внимание - я использовал два разных термина - внедрение и связка. Я сделал это умышленно, потому что есть разные способы взаимного существования JavaScript и HTML.

Самый простой из них - использование тега <SCRIPT>, который обычно расположен до тега <BODY> (принято размещать тег <SCRIPT> в теге <HEAD>), например:

```
<SCRIPT>
document.write("<h1>Привет, мир!</h1>");
</SCRIPT>
```

Данный сценарий будет выполнен при открытии страницы и выведет строку:

```
<h1>Привет, мир!</h1>
```

Однако не всегда нужно, чтобы ваша программа начинала работать сразу при открытии страницы. Чаще всего требуется, чтобы она запускалась при определенном событии, например при нажатии какой-то кнопки. При нажатии кнопки генерируется событие onClick. Обработчик для этого события можно указать прямо в HTML-коде, например:

```
<FORM>
<INPUT TYPE=button VALUE="Назад" onClick="history.back()">
</FORM>
```

Разумеется, не всегда обработчик события компактный. Иногда он просто не помещается в одну-две строчки. Конечно, максимальная длина значения атрибута HTML-тега составляет 1024 символа, что вполне хватило бы для несложных обработчиков. Однако такие длинные обработчики значительно ухудшают читабельность HTML-кода, поэтому если обработчик события достаточно длинный, его принято оформлять в виде функции, например:

```
<form>
<input type="button" value="Нажми меня"
onClick="openWindow()">
</form>
```

Функцию openWindow() нужно определить в теге <SCRIPT>. Полный код (HTML и JavaScript) нашего первого "проекта" (да, первая программа в этой книге не будет выводить фразу "Привет, мир!") приведен в листинге 2.1.

Листинг 2.1. Наш первый JavaScript-проект

```
<html>
 <head>
 <script language="JavaScript">
  function openWindow() {
     msgWindow= open("index.html")
   }
```

```
 </script>
 </head>
 <body>
  <form>
   <input type="button" value="Открыть главную страницу"
onClick="openWindow()">
  </form>
 </body>
</html>
```

Данная страница отобразит кнопку **Открыть главную страницу** (рис. 2.1), при нажатии которой будет открыто новое окно с загруженной страницей index.html.

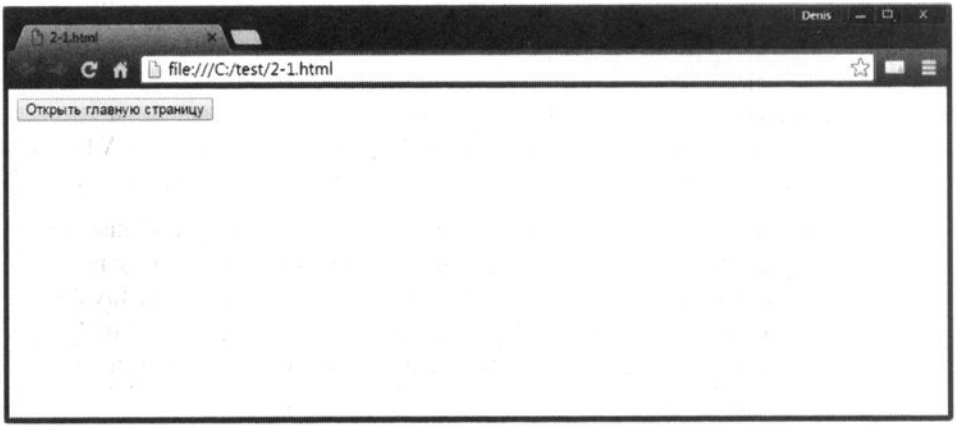


Рис. 2.1. Наша страница

Обратите внимание: мы указали атрибут language тега <SCRIPT>, указывающий язык программирования, на котором написан сценарий (кроме JavaScript сценарии можно писать еще на VBS – Visual Basic Script). Кроме атрибута language у тега <SCRIPT> есть еще один очень полезный атрибут – src, указывающий файл, в котором содержится JS-код. Если код JavaScript очень большой, и вы не хотите загромождать им HTML-код страницы, можете вынести его в отдельный файл, а затем подключить его следующим способом:

```
<script src="javascript.js" type="text/javascript"></script>
```

Четвертый (и последний) атрибут тега <SCRIPT> называется defer. Он откладывает выполнение сценария до полной загрузки страницы (по умолчанию сценарий начинает выполняться сразу при открытии страницы):

```
<script defer>
// code
</script>
```

Аргумент language, как уже было отмечено выше, задает язык программирования, на котором написан сценарий, и может принимать следующие значения:

- JavaScript – классический вариант языка программирования, который рассматривается в этой книге (разработка Netscape и Sun).
- JScript – разновидность языка программирования JavaScript, разработанная компанией Microsoft. Компании Microsoft обязательно нужно построить дом заново и заново изобрести велосипед, поэтому разница между JavaScript и JScipt заключается не только в названии, но и в подходах к программированию. Учтите, что поддержка JScript в некоторых браузерах может быть ограниченной, поэтому язык JavaScript более предпочтителен.
- VBS, VBScript – сценарий на языке программирования VBScript, в основе которого лежит Visual Basic. Поддерживается браузером Internet Explorer. Остальные браузеры или вовсе не поддерживают VBS или поддержка весьма ограничена.

В данной книге мы будем рассматривать классическую версию языка программирования, а именно – JavaScript. Значение атрибута language по умолчанию - JavaScript, поэтому если вы программируете именно на JavaScript, то можете не указывать атрибут language. Если же вы программируете на JScript, нужно обязательно указать атрибут language во избежание недопонимания со стороны браузера.

2.4. Комментарии в JavaScript

В этой главе вы уже успели познакомиться с комментариями, во всяком случае неявно. Теперь настало время познакомиться с ними явно. Комментарии в JS могут быть однострочными и многострочными. Однострочный комментарий начинается с двух знаков //, с этим типом комментариев вы уже знакомы:

```
// комментарий
i++;    // увеличиваем i
```

Комментарием является все, что находится после // и до конца строки.

Многострочный комментарий начинается символами /* и заканчивается символами */, например:

```
/* это пример
многострочного
комментария */
```

Какие комментарии использовать - зависит от вас. Однострочные комментарии удобно использовать для комментирования отдельных строчек кода. Многострочные подойдут для объяснения того, что делает целый блок, например, описать, что делает функция и какие параметры ей нужно передать.

2.5. Диалоговые окна

Для взаимодействия с пользователем, то есть для ввода данных и вывода результатов работы программы, как правило, используются HTML-формы и возможность вывода HTML-кода прямо в документ (метод document.write). Этот способ удобен тем, что вы, используя HTML и CSS, можете оформить форму ввода данных так, как вам заблагорассудится. То же самое можно сказать и о выводе данных. Из JS-сценария вы можете выводить любой HTML-код, позволяющий как угодно оформить вывод.

Но в некоторых ситуациях этих возможностей оказывается очень много. Иногда нужно просто вывести диалоговое сообщение, например сообщить пользователю, о том, что введенный им пароль слишком простой или слишком короткий. В этом разделе мы разберемся, как выводить диалоговые окна, позволяющие выводить короткие сообщения (например, сообщения об ошибках ввода) и обеспечивающие ввод данных.

2.5.1. Метод alert() - простое окно с сообщением и кнопкой OK

Метод alert() объекта window используется для отображения простого окна с сообщением и одной кнопкой - **OK**. Такое окно может использоваться, например, для отображения сообщений об ошибках (короткий/простой/неправильный пароль). Окно, кроме донесения до пользователя сообщения, больше не предусматривает никакого взаимодействия с ним.

Методу alert() передается только одна строка - отображаемая. Чтобы отобразить многострочное сообщение, разделяйте строки символом \n:

```
window.alert("Привет, мир!");
window.alert("Привет,\nмир!");
```

Первая наша программа была не "Hello, world", но давайте не будем изменять традиции и все-таки напишем эту программу, пусть она будет и не

первой - хоть тут мы будем отличаться от всех остальных программистов. Сценарий, демонстрирующий использование метода alert(), приведен в листинге 2.2.

Листинг 2.2. Использование метода alert()

```
<html>
<head>
  <title>Alert</title>
</head>
<body>
  <script>
    window.alert("Привет, мир!");
  </script>
</body>
</html>
```

Наш сценарий находится в теле документа (тег<body>), поэтому будет запущен сразу при загрузке HTML-файла. Изображаемое им окно приведено на рис. 2.2.

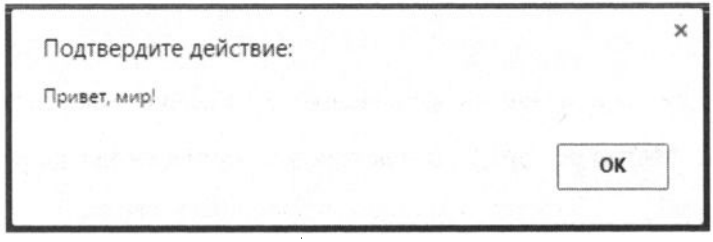


Рис. 2.2. Диалоговое окно в браузере Chrome

2.5.2. Метод confirm() - окно с кнопками OK и Cancel

Другой часто используемый метод - confirm(). Он выводит окно с сообщением и двумя кнопками - **OK** и **Cancel**, позволяя пользователю выбрать одну из них. Проанализировав возвращаемое методом значение (true, если нажата кнопка **OK** и false - в противном случае), вы можете выполнить то или иное действие. Для нашего примера мы будем просто выводить с помощью alert() название нажатой кнопки. Пример использования метода confirm() приведен в листинге 2.3.

Листинг 2.3. Использование метода confirm()

```
<html>
```

```
<head>
  <title>Confirm</title>
</head>
<body>
  <script>
   if (window.confirm("Нажмите OK или Отмена")) {
      window.alert("OK");
   }
   else {
      window.alert("Отмена");
   }
</script>
</body>
</html>
```

Результат работы этого метода приведен на рис. 2.3.

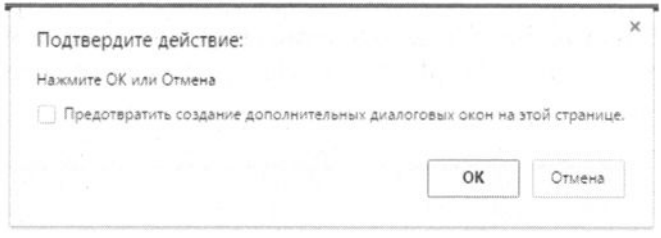


Рис. 2.3. Диалоговое окно, отображаемое методом confirm (браузер Firefox)

2.5.3. Метод prompt() - диалоговое окно для ввода данных

Метод prompt() отображает диалоговое окно с полем ввода, сообщением и кнопками OK и Cancel. Введенное пользователем значение можно потом будет присвоить какой-то переменной. Диалог возвращает введенную пользователем строку. Если пользователь ничего не ввел, диалог возвращает значение null.

Методу prompt() нужно передать два параметра - строку, которая будет отображена в качестве приглашения ввода (над полем для ввода данных), и значение по умолчанию, которое будет передано в сценарий, если пользователь поленится ввести строку и просто нажмет **OK**. Пример использования этого диалога приведен в листинге 2.4.

Листинг 2.4. Пример использования метода prompt()

```
<html>
<head>
  <title>Confirm</title>
```

```
</head>
<body>
  <script>
  var UName = window.prompt("Как тебя зовут?", "Никак");
  if (UName ==null) {
      alert("Пока!");
  }
  else {
      document.write("Привет, " + UName);
  }
</script>
</body>
</html>
```

Наш сценарий прост. Если пользователь нажмет **Отмена**, то увидит диалог с текстом "пока!" (ну не хотим мы общаться с пользователем, который не хочет представиться). Если пользователь нажмет **OK**, то строка из поля ввода будет отображена в HTML-документе в виде "Привет, <введенный текст>".

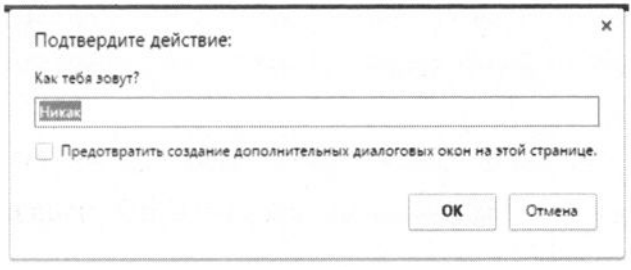


Рис. 2.4. Метод prompt() в действии (браузер Firefox)

2.6. Специальные символы

В строках вы можете использовать специальные символы. Это комбинации обычных символов, обозначающих служебные или непечатаемые символы, которые нельзя ввести обычным способом. Например, с помощью специального символа \n в строку можно вставить разрыв строки, что позволяет разбить сообщение в диалоговом окне на строки, например:

```
window.alert("Ошибка!\nПароль неправильный");
```

В JS вы можете использовать следующие специальные символы (ради справедливости, нужно отметить, что их можно также использовать в языках C, PHP и некоторых других):

- \n - перевод строки;

- \r - возврат каретки (в современном программировании используется очень редко);
- \t - табуляция;
- \f - перевод страницы;
- \' - апостроф;
- \" - двойная кавычка;
- \\ - обратный слеш (косая черта).

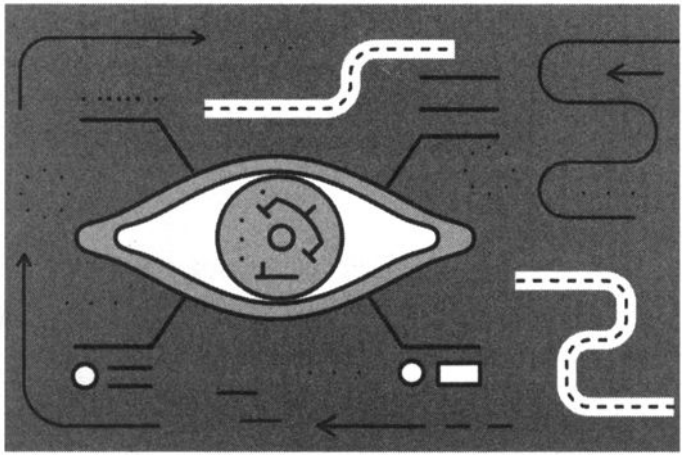
2.7. Зарезервированные слова

Зарезервированные слова (ключевые слова) нельзя использовать в качестве идентификаторов, то есть имен переменных, функций и объектов. Зарезервированные слова JavaScript приведены в таблице 2.1.

Таблица 2.1. Зарезервированные слова JavaScript

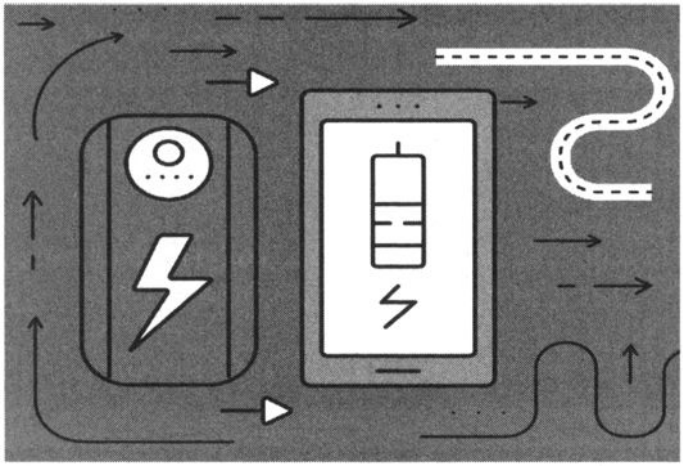
abstract	extends	interface	synchronized
boolean	false	long	this
break	final	native	throw
byte	finally	new	throws
case	float	null	transient
catch	for	package	true
char	function	private	try
class	goto	protected	var
const	if	public	void
continue	implements	return	while
default	import	short	with
do	in	static	
double	instanceof	super	
else	int	switch	

На этом наша вводная в JavaScript глава заканчивается. В следующей мы рассмотрим синтаксис JavaScript. Большая часть материала в этой главе воспринималась интуитивно. Вы всё понимали, но все равно у вас есть вопросы относительно синтаксиса языка программирования. Конкретизировать, казалось бы, явные вещи и предназначена глава 3. В ней мы рассмотрим переменные, операторы и многое другое.



Глава 3.

Основы синтаксиса



3.1. Переменные в JavaScript

3.1.1. Объявление переменной

Переменная – это поименованная область памяти, хранящая данные. В других языках программирования (например, в C, PHP) переменные являются типизированными, то есть при объявлении переменной определяется тип данных (число, символ, строка, массив чисел или массив символов и т.д.), которые будут храниться в этой переменной. В JavaScript, как и в PHP, переменные не являются строго типизированными, а это означает, что тип переменной зависит от данных, которые в данный момент хранятся в ней.

В отличие от того же PHP, где можно использовать переменную без ее предварительного объявления (хотя это и нежелательно, поскольку переменная не инициализирована), в JavaScript переменную нужно объявить с помощью служебного слова var (сокращение от variable), как во многих других языках программирования (например, предварительное объявление переменных требуется в C, Pascal и других языках программирования).

Имя переменной должно начинаться с символа буквы (A-Z) или символа подчеркивания. Последующими символами могут быть цифры, буквы, а также знак $. Имя переменной не может начинаться с цифры или с символа $ (как в PHP)! Также нужно помнить, что JavaScript учитывает регистр символов, то есть переменные variable и Variable – это две разные переменные.

Имя переменной не может быть зарезервированным словом. Зарезервированные слова JavaScript были приведены в таблице 2.1.

Правильные примеры имен переменных:

```
x, y1, userName, _user
```

Неправильные имена:

```
1x, public
```

В первом случае имя переменной начинается с цифры, во втором - является зарезервированным словом.

При объявлении переменной желательно указать ее первоначальное значение (инициализировать ее):

```
var my = 1;
```

Можно объявить переменную и без инициализации, однако это нежелательно, поскольку если забыть ее инициализировать перед первым использованием, результаты вычисления могут быть неопределенными.

При желании можно объявить сразу несколько переменных, разделив их запятыми:

```
var x, y1, userName, _user;
```

Сейчас переменная first целого типа, поскольку содержит значение 1. Но ее очень легко превратить в строку, присвоив строковое значение:

```
my = "привет";
```

Все последующие обращения к переменной производятся без служебного слова var.

3.1.2. Типы данных и преобразование типов

Данные, хранящиеся в переменной, могут быть разного типа. Как вы заметили, при объявлении переменной (как это делается в других языках программирования) тип переменной (данных) не указывается.

В JavaScript переменные могут содержать следующие типы данных:

- number - числа, как целые, так и с плавающей точкой.
- string - строки.
- Boolean - логический тип данных, может содержать два значения - true (истина) и false (ложь).
- function - функции. В JS мы можем присвоить ссылку на функцию любой переменной, если указать имя функции без круглых скобок.
- object - массивы, объекты, а также переменные со значением null.

Тип переменной JavaScript определяет при ее инициализации, то есть при первом присваивании значения, например:

```
Num1 = 5;    // Переменной Num1 присвоено целое значение 5, тип
- number
Num2 = 5.5;     // Переменное с плавающей точкой 5.5, тип -
number
Str1 = "привет";  // Переменной Str1 присвоено значение
"Hello", тип - string
Str2 = 'мир';  // Также можно использовать одинарные кавычки
Str3 = null; // Переменная не содержит данных, ее тип - object
Bool1 = true;  // Булевая (логическая) переменная со значением
true
```

Оператор typeof возвращает строку, описывающую тип данных переменной. Давайте продемонстрируем его работу (см. листинг 3.1). Сценарий из листинга 3.1 объявляет переменные, выполняет их инициализацию (присваивает значения), а затем выводит тип каждой переменной. Результат работы этого сценария изображен на рис. 3.1.

Листинг 3.1. Оператор typeof

```
<html>
<head>
  <title>typeof</title>
</head>
<body>
<script>
var Num1, Num2, Str1, Str2, Str3, Bool1;

Num1 = 5;
Num2 = 5.5;
Str1 = "привет";
Str2 = 'мир';
Str3 = null;
Bool1 = true;

document.write("<br>Num1 - " + typeof(Num1));
document.write("<br>Num2 - " + typeof(Num2));
document.write("<br>Str1 - " + typeof(Str1));
document.write("<br>Str2 - " + typeof(Str2));
document.write("<br>Str3 - " + typeof(Str3));
document.write("<br>Bool1 - " + typeof(Bool1));

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

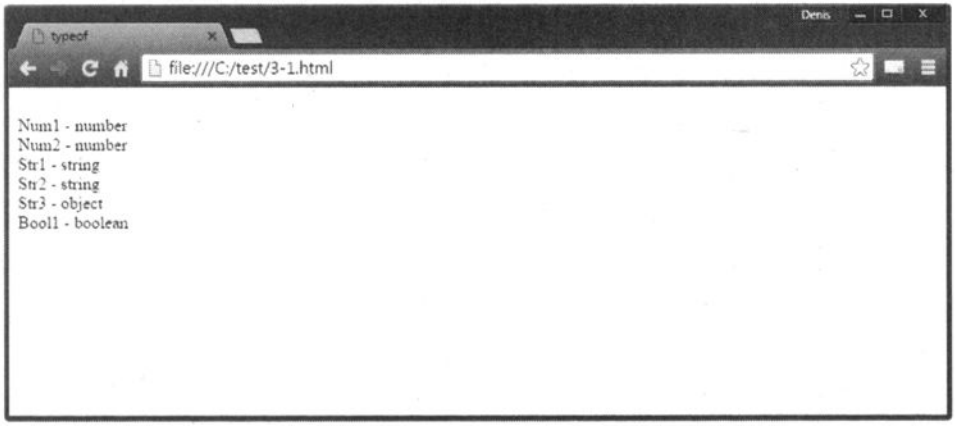


Рис. 3.1. Пример использования оператора typeof

В процессе вычислений JavaScript производит преобразование типов. Посмотрим, как оно работает. Определим две переменные: одна будет содержать число 1, а вторая – символ "1":

```
var A = 1;
var B = "1";
```

Теперь определим еще две переменные:

```
var SR = B + A;      // string result
var IR = A + B;      // integer result
```

Тип переменной определяется по типу первого присваиваемого значения. В нашем случае переменная SR будет содержать значение "11", поскольку первой была строковая переменная B. Переменная IR будет содержать значение 2 по вышеописанным причинам.

Переменные в JavaScript также могут быть булевого (логического) типа. Такие переменные могут принимать одно из двух значений – true (истина) или false (ложь):

```
var Bool = true;
```

Для принудительного преобразования типов вы можете использовать две следующие функции:

1. parseInt - преобразует строку в целое число, если это возможно.
2. parseFloat - преобразует строку в число с плавающей запятой, если это возможно.
3. eval - вычисляет выражение в строке, как будто это обычное выражение JavaScript.

Рассмотрим несколько примеров:

```
var A = "1";   // строковое значение "1"
var B = parseInt(A);      // переменная B теперь содержит число
1
var C = "5.1";    // строка "5.1"
var D = parseFloat(C);    // переменной D будет присвоено число
5.1
var E = "2 + 2";     // строка "2+2"
var F = eval(E);     // переменной F будет присвоено число 4
```

3.1.3. Локальные и глобальные переменные

Как и в других языках программирования, в JavaScript существуют локальные и глобальные переменные. Локальной называется переменная, объявленная в какой-нибудь функции. Она доступна только в этой функции и недоступна во всем скрипте.

Глобальная переменная объявлена в теле скрипта и доступна во всех объявленных в скрипте функциях.

Глобальные переменные принято объявлять в самом начале скрипта, чтобы все функции наследовали эти переменные и их значения. Подробнее об области видимости переменной мы поговорим в главе 11, когда будем рассматривать функции.

3.2. Выражения и операторы

3.2.1. Типы выражений

Выражение – это набор переменных, констант, операторов. У любого выражения есть свое значение – результат вычисления выражения. Значение может быть числом, строкой или логическим значением.

В JavaScript есть два типа выражений: которые присваивают значение переменной и те, что просто вычисляют выражение без присваивания его значения переменной:

```
x = 3 * 2

9 - 5
```

Существуют еще так называемые условные выражения. Они определяются так:

```
(условие)? значение1: значение2
```

Если условие истинно, то выражение имеет значение 1, а если нет – значение 2.

Например:

```
sedan = (doors >=4) ? true : false
```

3.2.2. Операторы присваивания

Операторы присваивания, поддерживаемые в JavaScript, описаны в таблице 3.1.

Таблица 3.1. Операторы присваивания

Оператор	Пример	Описание
+=	x += y	x = x + y
-=	x -= y	x = x – y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%= (остаток от деления)	x %= y	x = x % y

3.2.3. Арифметические операторы

Математические (арифметические) операторы в JS такие же, как и в большинстве других языков программирования, а именно:

- + - сложение (например, A = B + C).

- - - вычитание (например, A = B - C).
- * - умножение (например, A = B * C).
- / - деление (например, A = B / C).
- % - деление по модулю (например, A = B % C).
- ++ - инкремент, увеличивает значение переменной на 1 (например, i++).
- -- - декремент, уменьшает значение переменной на 1 (например, j--).

Как используются эти операторы, думаю, вы и так знаете. Нужно отметить только особенность операторов инкремента и декремента. Рассмотрим небольшой пример:

```
x = 7
y = x++
```

Переменной y будет присвоено значение 7, а после этого переменная x будет увеличена на 1 (значение 8). Если же ++ указать до x (а не после него), то сначала переменная x будет увеличена на 1, а потом уже будет присвоено новое значение переменной y:

```
y = ++x
```

3.2.4. Логические операторы

К логическим операторам относятся следующие операции:

- ! – унарная операция отрицания (NOT).
- && - бинарная операция И (AND), истинна, когда оба операнда истинны.
- || - бинарная операция ИЛИ (OR), истинна, когда один из операндов равен true.

Пример:

```
bool = true && false
```

3.2.5. Операторы сравнения

В таблице 3.2 описаны операторы сравнения, которые вы можете использовать в JavaScript при написании своих сценариев.

Таблица 3.2. Операторы сравнения в JavaScript

Оператор	Описание
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно
!=	Не равно
===	Строго равно

Обратите внимание, что в JS есть два оператора сравнения: равно (==) и строго равно (===). В чем между ними разница? Оператор ==, сравнивая значения разных типов, пробует свести типы, а затем выполнить сравнения. То есть значение 3 будет равно строке "3". Оператор ===, встретив разные типы значений, сразу вернет false.

3.2.6. Двоичные операторы

К двоичным операторам относятся следующие операторы:

- ~ - двоичная инверсия (A = ~B).
- & - двоичное И (A = B & C).
- | - двоичное ИЛИ (A = B | C).
- ^ - двоичное исключающее ИЛИ (A = B ^ C).
- << - сдвиг влево на один или более разрядов с заполнением младших разрядов нулями (A = B << Y).
- >> - сдвиг вправо на один или разрядов с заполнением старших разрядов содержимым самого старшего разряда (A = B >> Z).
- >>> - сдвиг вправо без учета знака, старшие разряды будут заполнены нулями (A = B >>> C).

Двоичные операторы выполняют поразрядные операции с двоичным представлением целых чисел.

3.2.7. Слияние строк

Для слияния (конкатенации) строк используется оператор +:

```
var str = "string1" + "string2"
```

В результате переменная str будет содержать значение "string1 string2".

3.2.8. Приоритет выполнения операторов

Сейчас мы поговорим о приоритете выполнения операторов. Пусть у нас есть выражение:

```
A = 2 + 3 * 4 / 5;
```

В какой последовательности будет производиться его вычисление? Еще из школьного курса математики мы знаем, что приоритет операции умножения выше, чем сложения, поэтому сначала 3 будет умножено на 4 (в результате мы получим значение 12).

Затем полученное значение 12 будет разделено на 5 (мы получим значение 2.4), поскольку приоритет операции деления выше, чем сложения.

К полученному значению 2.4 будет добавлено значение 2 и в результате переменной A будет присвоено значение 4.4.

Приоритет операций можно изменить с помощью скобок, например:

```
A = (2 + 3) * 4 / 5;
```

В этом случае сначала будет вычислено выражение 2 + 3, а затем полученное значение будет умножено на 4 и разделено на 5. Приоритет операций умножения и деления одинаковый, поэтому они выполняются слева направо, то есть *в порядке следования*. В результате будет получено значение 4 (5 * 4 и разделить на 5).

Далее перечислены операторы в порядке убывания приоритета:

- !, ~, ++, -- — отрицание, двоичная инверсия, инкремент, декремент.
- *, /, % — умножение, деление, остаток от деления.
- +, - — сложение и вычитание.
- <<, >>, >>> — двоичные сдвиги.
- & — двоичное И.

- ^ — двоичное исключающее ИЛИ.
- | — двоичное ИЛИ.
- =, +=, -=, *=, /=, %= — присваивание.

3.3. Основные конструкции языка

К основным конструкциям языка относят условный оператор (if..else), а также операторы циклов. В этом разделе будут рассмотрены данные конструкции.

3.3.1. Условный оператор if

Прежде чем мы будем рассматривать условный оператор if, настоятельно рекомендую вернуться к разделу 3.2.5 и еще раз просмотреть таблицу с операторами сравнения - так вам будет понятнее все происходящее здесь.

Условный оператор if имеется в большинстве языков программирования. Он позволяет выполнить определенное действие в зависимости от истинности условия. Общая форма оператора выглядит так:

```
if ( условие)
{операторы, если условие истинно}
[else {
операторы, если условие ложно}]
```

Обратите внимание, что вторая часть (else) не обязательна.

Условие - это логическое выражение, построенное на базе операторов сравнения, именно поэтому я просил вас вернуться к разделу 3.2.5, чтобы еще раз просмотреть имеющиеся операторы сравнения. Каждый из операторов сравнения возвращает true в случае истинности и false, если проверяемый факт ложен.

Пусть у нас есть две переменные:

```
var A = 10;
var B = 5;
```

Оператор A == B вернет false, поскольку A не равно B. Оператор A > B вернет true, поскольку A больше, чем B.

Для инверсии логического значения вы можете использовать оператор !, например:

```
!(A == B)
```

Конечно, можно также использовать оператор !=, но здесь уже поступайте, как вам будет удобнее и понятнее.

В нашем случае переменные A и B не равны, поэтому оператор == вернет значение false, но поскольку указан оператор !, то будет возвращено значение true.

Рассмотрим несколько примеров:

```
var A = 10;
var B = 5;

// Будет выведено A > B
if (A > B) {
   document.write('A > B'); }

// Будет выведено B < A
if (A > B) {
   document.write('A > B'); }
else {
   document.write('B < A'); }

// Будет выведено A = B
if (!(A == B)) {
   document.write(" A != B "); }
else {
   document.write(" A = B "); }
```

В главе 8 мы уже сталкивались с оператором if и рассмотрели более сложный пример - проверку нажатия одной из кнопок в диалоговом окне. Вы можете вернуться к этому примеру, чтобы освежить его в памяти.

Операторы if можно вкладывать друг в друга, что продемонстрировано в листинге 3.2. Сценарий в этом листинге пытается разделить 10 на значение одной из переменных - A или B, предварительно проверяя, не равно ли значение этих переменной 0. Сначала он выясняет, не равно ли 0 значение переменной A. Поскольку A = 0, то выполнение сценария переходит на второй оператор if, который проверяет, не равно ли 0 значение переменной B. С переменной B все хорошо, поэтому переменной C будет присвоено зна-

чение 10 / 1 (10). В противном случае, если обе переменные равны 0, будет выведено сообщение Division by zero.

Листинг 3.2. Вложенность операторов if

```
<html>
<head>
  <title>if</title>
</head>
<body>
<script>

var A = 0;
var B = 1;
var C;

if (A !=0) {
   C = 10 / A;
   document.write('C = ' + C); }
else if (B != 0) {
   C = 10 / B;
   document.write('C = ' + C); }
else document.write("Деление на 0");

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

3.3.2. Оператор выбора switch

Иногда (конечно не во всех ситуациях) вместо множества вложенных операторов if можно использовать оператор switch. Оператор switch позволяет сравнить переменное или выражение с множеством значений, что позволяет избавиться от серии операторов if и сделать код более компактным.

Общая форма оператора switch выглядит так:

```
switch (<Переменная или выражение>) {
case <Значение 1>:
<Оператор 1>;
break;
case <Значение 2>:
<Оператор 2>;
break;
...
[default:
```

```
<Оператор>;]
}
```

Работает оператор switch следующим образом:

- Сначала вычисляется значение переменной или выражения.
- Затем полученное значение сравнивается с одним из значений, указанных в блоках case.
- Представим, что у нас 10 блоков case, и значение совпало с 5-м блоком case. Тогда, если в 5-м блоке case не указан оператор break, то будут выполнены действия, связанные с блоками 5-10, а также операторы из блока default. Если же указан оператор break, тогда будет выполнено только то действие, которое указано в 5-м блоке case. Для большей однозначности (если не нужно иного) я всегда рекомендую использовать оператор break для преждевременного выхода из оператора switch.
- Если вычисленное значение не совпало ни с одним из значений, указанных в блоках case, тогда будет выполнены операторы из блока default, если таковой указан. Блок default является необязательным.

Представим, что у нас есть переменная command, в зависимости от значения которой нужно выполнить определенные действия, например:

```
if (command == 1) alert('Выбрано действие: 1');
if (command == 2) alert('Выбрано действие: 2');
if (command == 3) alert('Выбрано действие: 3');
if (command == 4) alert('Выбрано действие: 4');
```

Код выглядит громоздко и логически воспринимается не как один блок, а как четыре разных блока (если бы мы по этому коду построили блок-схему, то у нас бы и получилось четыре разных блока).

Весь этот громоздкий код мы можем заменить на более компактный. Пусть он занимает больше строк, зато выглядит не таким перегруженным и воспринимается как единое целое:

```
switch (command) {
   case 1: alert('Chosen action: 1'); break;
case 2: alert('Chosen action: 2'); break;
case 3: alert('Chosen action: 3'); break;
case 4: alert('Chosen action: 4'); break;
default: alert('Unknown action!);
}
```

Как видите, получившийся код воспринимается не таким перегруженным, хотя занимает больше строк. К тому же оператор switch позволяет задавать действие по умолчанию. В конечном итоге, с его помощью можно понятнее и прозрачнее реализовывать сложные разветвления, которые кажутся запутанными, если их реализовать с помощью if. Однако еще раз отмечу, что switch - далеко не панацея во всех ситуациях выбора.

В листинге 3.3 приведен пример использования оператора switch. Сначала мы отображаем диалог ввода действия, затем анализируем, какое действие выбрал пользователь. Обратите внимание, что прежде чем передать полученное действие оператору switch, мы сводим его к типу number с помощью функции parseInt().

Листинг 3.3. Пример использования оператора switch

```
<html>
<head>
  <title>Confirm</title>
</head>
<body>
  <script>
  var command = window.prompt("Введите действие", "");

  if (command == null) {
    document.write('Cancel pressed'); }
   else
     switch (parseInt(command)) {
        case 1: alert('Выбрано действие 1'); break;
        case 2: alert('Выбрано действие 2'); break;
        case 3: alert('Выбрано действие 3'); break;
        case 4: alert('Выбрано действие 4'); break;
        default: alert('Неизвестное действие');
     }

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

3.3.3. Циклы

Если проанализировать все программы, то на втором месте после условного оператора будут операторы цикла. Используя цикл, вы можете повторить операторы, находящиеся в теле цикла. Количество повторов зависит от

типа цикла - можно даже создать бесконечный цикл. В JavaScript есть три типа цикла:

- Цикл со счетчиком (for).
- Цикл с предусловием (while).
- Цикл с постусловием (do..while).

Цикл for

Начнем с цикла со счетчиком - for. Он используется для выполнения тела цикла четко определенного количества раз. Цикл while, например, удобно использовать для ожидания какого-то события (мы не знаем, сколько раз будет выполнено тело цикла, пока условие станет истинным), а цикл for используется тогда, когда вы точно знаете, сколько раз нужно повторить цикл.

Синтаксис цикла **for**:

```
for (команды_инициализации; условие; команды_после_итерации) {
<Операторы - тело цикла>
}
```

Оператор **for** начинает свою работу с выполнения команд инициализации. Данные команды выполняются всего лишь один раз. После этого проверяется условие: если оно истинно, выполняется тело цикла. После того, как будет выполнен последний оператор тела, выполняются команды "после итерации". Затем снова проверяется условие, в случае, если оно истинно, выполняются тело цикла и поститерационные команды и т.д.

Выведем строку 0123456789:

```
for (i=0; i<10; i++) document.write(i);
```

Чтобы вывести строку 12345678910, нужно установить начальное значение счетчика в 1 и изменить условие цикла:

```
for (i=1; i<=10; i++) document.write(i);
```

Цикл for будет очень полезным при обработке массивов, которые мы рассмотрим в следующей главе.

Цикл while

В некоторой мере цикл for очень похож на цикл с предусловием (while), так как сначала проверяется условие, а потом уже выполняется тело цикла. Рассмотрим цикл с предусловием:

```
while ( логическое выражение ) {
 операторы; }
```

Сначала цикл вычисляет значение логического выражения. Если оно истинно, происходит итерация цикла (выполняется тело цикла), иначе следует выход из цикла и выполнение следующего за циклом оператора.

Далее приведен пример, выводящий числа от 1 до 10:

```
var i = 1;

while (i < 11) {
document.write(i + "<br>");
i++;
}
```

Соблюдайте осторожность при использовании цикла while. Если вы не предусмотрите условие выхода, получите бесконечный цикл. В нашем случае условием выхода является не только, что i < 11, но и сам инкремент i, то есть оператор, способный повлиять на условие, указанное в заголовке цикла. Если бы мы забыли указать оператор i++, то получили бы бесконечный цикл. Поскольку переменная i = 1, что меньше 11, и у нас нет другого оператора, который в теле цикла изменял бы эту переменную, тело цикла будет выполняться бесконечно.

Цикл do..while

В JavaScript есть еще одна форма цикла - do while. В отличие от цикла while здесь сначала выполняются операторы (тело цикла), а затем уже проверяется условие. Если условие истинно, то начинается следующая итерация. Получается, что тело цикла будет выполнено как минимум один раз. Синтаксис цикла:

```
do
 {
 // тело цикла
 }
while (условие);
```

Пример:

```
i = 1;
do {
  document.write(i);
  i++; }
while (i < 10);
```

Операторы break и continue

В теле цикла вы можете использовать операторы break и continue. Первый прерывает выполнение цикла, а второй – выполнение текущей итерации и переходит к следующей.

Представим, что нам нужно вывести только нечетные числа в диапазоне от 1 до 20. Пример цикла может быть таким:

```
for (i=1; i<21; i++) {
 if (i % 2 == 0) continue;
 else document.write(i + " ");
}
```

В теле цикла мы проверяем, если остаток от деления i на 2 равен 0, значит, число четное и нужно перейти на следующую итерацию цикла. В противном случае нам нужно вывести наше число.

А вот пример использования оператора break. Несмотря на то, что цикл якобы должен выполниться 10 раз, он будет прерван, когда i будет равно 5:

```
for (i=1; i<11; i++) {
 if (i == 5) break;
 document.write(i + " ");
}
```

В результате будет выведена строка:

```
1 2 3 4
```

Вложенность циклов

В теле цикла может быть другой цикл. Вложенность циклов формально не ограничивается, однако нужно быть предельно осторожным, чтобы не допустить зацикливания. Пример вложенного цикла:

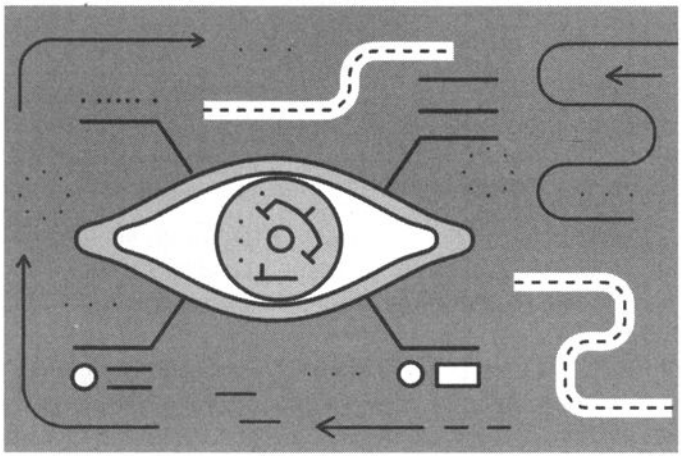
```
var j = 1;
while (j < 15) {
 for (k=0; k<j; k++) document.write('*');
 document.write('<br>');
 j++;
}
```

Результат выполнения этого сценария приведен на рис. 3.2.

Одно из частых применений циклов - обработка массивов. О том, что такое массивы и как с ними работать в JavaScript, мы поговорим в следующей главе.

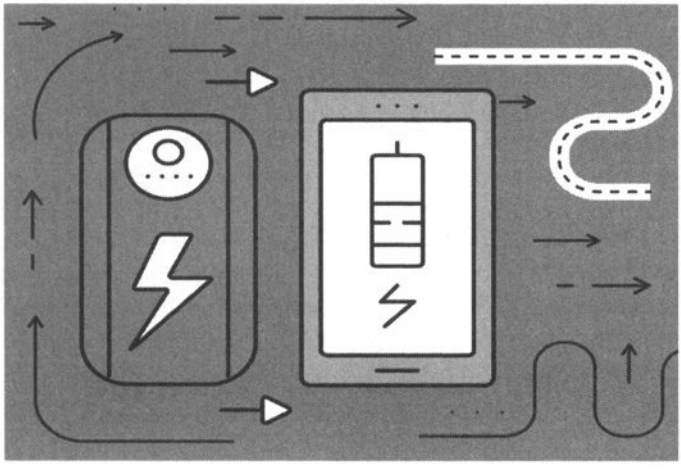
Рис. 3.2. Пример работы вложенных циклов


JavaScript™



Глава 4.

Массивы



4.1. Введение в массивы

Все мы знаем, что такое переменная, - это поименованная область в памяти. Мы выделяем область в памяти (точнее, за нас это делает интерпретатор) и назначаем ей имя (а вот имя выбираем мы - программисты). Но переменные не всегда удобны. Представим, что нам нужно хранить набор данных, состоящий из 100 или даже 1000 значений одного типа. В этом случае нам понадобится 100 или 1000 переменных, что не совсем удобно. Специально для таких целей были созданы массивы. Массив - это упорядоченный набор данных. У каждого элемента массива есть свой индекс (его также называют ключом), который используется для однозначной идентификации элемента внутри массива. Думаю, совершенно понятно, что в одном массиве не может быть двух элементов с одинаковыми ключами.

Работать с массивами намного удобнее, поскольку можно в цикле перебрать все элементы массива, а не обращаться отдельно к каждому элементу (как это происходит с переменными).

Если вы программировали на других языках программирования, то знаете, что все переменные, в том числе и массивы, должны быть объявлены в строго определенном месте программы. В JavaScript переменную, следовательно и массив, можно объявить в любом месте программы (сценария), но до первого использования массива. А это удобно, если нужно создать массив с неизвестным числом параметров. Например, вам нужно прочитать в массиве текстовый файл так, чтобы каждая строка файла представляла собой один элемент массива. В другом языке программирования нужно было объявить очень большой массив, скажем, на 10 000 элементов, - ведь вы же не знаете, сколько строк будет в файле, но нужно написать более или менее универсальную программу, чтобы она умела обрабатывать большие файлы. А что делать, если в файле всего 3 строки или 10 001 строка? В первом случае вы выделили памяти во много раз больше, чем реально нужно для обработки файла. Ваша программа будет попросту поедать память, и вы ничего не сможете сделать. Во втором случае ваша программа не справится с обработкой всего файла, поскольку в файле больше строк, чем может поместиться в массив.

4.2. Инициализация массива

Массив инициализируется, как и любая другая переменная, путем присваивания значения. Поскольку массив может содержать несколько значений, при его инициализации значения указываются в квадратных скобках и разделяются запятыми:

```
var M;
M = [5, 3, 4, 1, 2];

var Months;
Months = ["", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
"Aug", "Sep", "Oct", "Nov", "Dec"];
```

Нумерация элементов массива начинается с 0, а не с 1. Получить доступ к определенному элементу массива можно так:

```
num0 = M[0];   // 5
j = M[3];    // 1
```

4.3. Изменение и добавление элементов массива

При желании вы всегда можете изменить значение существующего элемента массива или добавить новый элемент, например:

```
M[0] = 7;
M[5] = 8;
```

Первый оператор присваивает значение 7 элементу массива M с индексом 0. Второй оператор создает новый элемент массива со значением 8, индекс элемента - 5. Теперь у нас есть вот такой массив:

```
[7, 3, 4, 1, 2, 8]
```

Посмотрим, что сделает следующий оператор:

```
M[7] = 11;
```

Этот оператор создаст два элемента массива: элемент с индексом 6 и значением undefined и элемент с индексом 7 и значением 11. Массив будет следующим:

```
[7, 3, 4, 1, 2, 8, undefined, 11]
```

4.4. Многомерные массивы

В JS вы можете создавать многомерные массивы путем присваивания любому элементу массива нового массива, например:

```
M[0] = [3, 2, 1];
```

Обратиться к элементу многомерного массива можно так:

```
j = M[0][1];
```

4.5. Пример обработки массива

Подробно о средствах работы с массивами мы поговорим в главе 14, когда будем рассматривать встроенные классы JS. Сейчас же мы рассмотрим несколько примеров для работы с массивами. У любого объекта массива есть свойство length, содержащее длину массива. Это свойство мы можем использовать при обработке массива.

Сценарий из листинга 4.1 вычисляет минимальный элемент массива. Сначала минимальным считается первый (с индексом 0) элемент массива. Далее в цикле for мы сравниваем каждый следующий элемент массива. Если сравниваемый элемент меньше нашего минимума, он становится новым минимумом.

Листинг 4.1. Вычисление минимума массива

```
<html>
<head>
  <title>Минимум в массиве</title>
</head>
```

```
<body>
  <script>
  var M = [7, 66, 55, 4, 88, 1, 8, 99, 3];

  Min = M[0];    // Минимум
  Min_ind = 0;   // Индекс Min

  document.write('Массив: <br>');

  for (i=1; i<M.length; i++) {
    if (M[i] < Min) {
      Min = M[i];
      Min_ind = i;
     } // if
     document.write(M[i] + " ");
  } //for
  document.write('<br>Минимум: ' + Min);
  document.write('<br>Индекс: ' + Min_ind);

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

Сначала мы считаем минимальным элемент с индексом 0. Далее выводим этот минимальный элемент, поскольку обработка массива начинается с индекса 1 (сравнивать 0-й элемент с 0-м элементом нет смысла). В цикле мы проверяем, не является ли текущий элемент минимальным, и, если это так, устанавливаем новый минимум. Для идентификации минимума используются две переменных - Min (минимум) и Min_ind (индекс минимума). Также в цикле мы выводим обрабатываемый элемент. Результат работы сценария приведен на рис. 4.1.

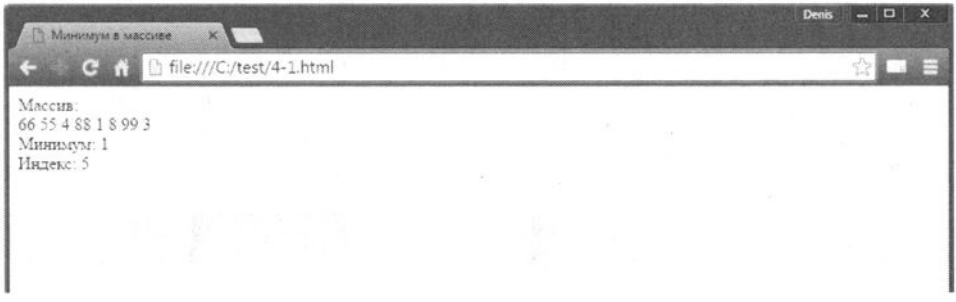


Рис. 4.1. Поиск минимального элемента

Теперь мы решим другую подобную задачу - найдем максимальный элемент. Чтобы было интереснее, давайте сделаем двумерный массив и найдем максимальный элемент в нем. А то наш новый сценарий отличался бы от предыдущего лишь знаком больше. Согласитесь, совсем не интересно. В новом же сценарии нам нужно обойти два измерения. Для обхода первого измерения мы будем использовать счетчик i, для обхода второго - j. Как только найдем максимальный элемент, в массив Ind мы запишем его "координаты" - значения счетчиков i (элемент 0) и j (элемент 1).

Листинг 4.2. Поиск максимума в двухмерном массиве

```
<html>
<head>
  <title>Максимальный элемент</title>
</head>
<body>
  <script>
  var M = [0, 0, 0];

  M[0] = [3, 2, 1];
  M[1] = [7, 8, 9];
  M[2] = [5, 6, 7];

  Max = M[0][0];    // Максимум
  Ind = [0, 0];     // Его индекс

  for (i=0; i<M.length; i++) {

    for (j=0; j<M[i].length; j++) {

      if (M[i][j] > Max) {
      Max = M[i][j];
      Ind[0] = i;
      Ind[1] = j;

      }

    }

  }

  document.write('Max ' + Max);
  document.write('<br>Ind [' + Ind[0] + '][' + Ind[1] + ']');

</script>
```

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

Обратите внимание, как мы обращаемся к свойству length. Поскольку каждый из элементов исходного массива также является массивом, то мы можем обратиться к length так: M[i].length. Конечно, мы и так знаем размерность массива, и можно было бы использовать значения длины 3 для каждого из счетчиков, но корректнее. Результат работы сценария 4.2 изображен на рис. 4.2.

Хватит теории. Настало время для практики, и в следующей главе мы разработаем простенький слайдер для нашего сайта.

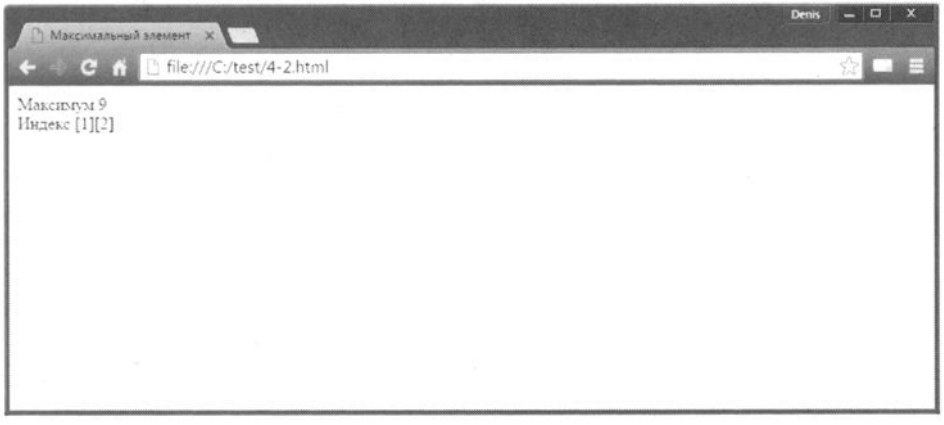
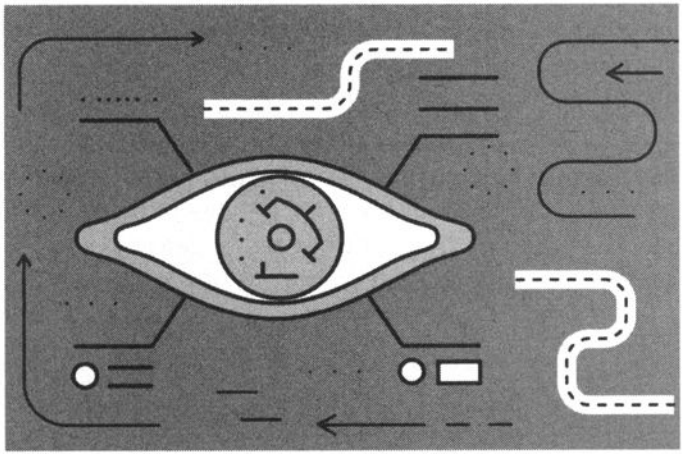


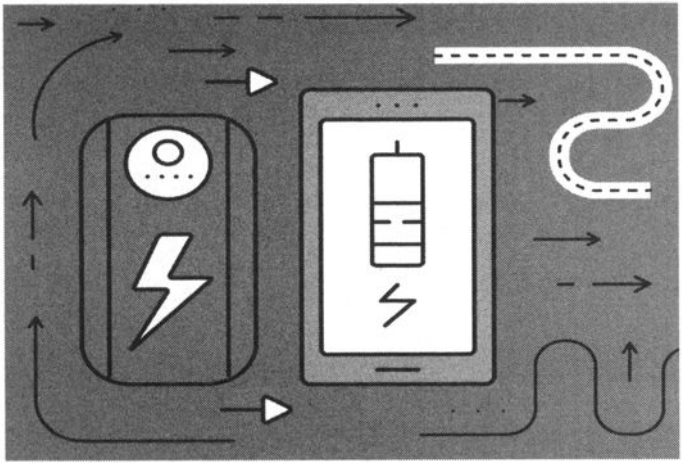
Рис. 4.2. Поиск максимума в двухмерном массиве


JavaScript™



Глава 5.

Делаем слайдер



Слайдер - это средство отображения изображений в виде слайдов. Обычно слайдеры имеют возможность как автоматического слайд-шоу (когда слайды показываются автоматически), так и ручного переключения слайдов. В этой главе мы попытаемся организовать слайдер вручную, а затем покажем, как его можно сделать с помощью jQuery UI.

5.1. Делаем слайдер вручную

Код простейшего слайдера довольно компактный. Нам нужно предусмотреть массив с изображениями, функции инициализации слайдера, функции прокрутки слайдера влево и вправо, а также автоматическое перелистывание картинок. Такой код приведен в листинге 5.1.

Листинг 5.1. Код слайдера

```
var slider = {
      // массив с изображениями
   slides:['6.jpg','9.jpg','20.jpg'],
   // начальный кадр (индекс из массива, нумерация с 0)
   frame:0,
   set: function(image) { // установка нужного фона слайдеру
      document.getElementById("scr").style.backgroundImage =
"url("+image+")";
   },
   init: function() { // запуск слайдера с начальной картинкой
      this.set(this.slides[this.frame]);
   },
   left: function() { // влево
      this.frame--;
      if(this.frame < 0) this.frame = this.slides.length-1;
      this.set(this.slides[this.frame]);
   },
   right: function() { // вправо
      this.frame++;
      if(this.frame == this.slides.length) this.frame = 0;
      this.set(this.slides[this.frame]);
   }
};
window.onload = function() { // запуск слайдера после загрузки
документа
```

```
   slider.init();
   setInterval(function() { // 5 секунд
      slider.right();       // после - переход на кадр справа
   },5000);
};
```

HTML-код страницы выглядит так:

```
<div id="main">
   <button class="left" onclick="slider.left();"><</button>
   <div id="scr"></div>
   <button class="right" onclick="slider.right();">></button>
</div>
```

Конечно, чтобы слайдер имел презентабельный вид, нужно задать стили. Стили, а также полный код примера приведены в листинге 5.2.

Листинг 5.2. Полный код слайдера

```
<html lang="ru">
<head>
   <title>Слайдер</title>
   <meta charset="utf-8">
</head>
<style>
#main {
   position:relative;
   margin: 100px auto;
   padding: 5px;
   width: 660px;
   height:360px;
   background-color: silver;
   border: 5px solid grey;
   border-radius:15px;
}
#scr {
   margin:20px auto;
   width: 600px;
   height: 320px;
   margin-top:20px;
   background-color: white;
   background-size:cover;
   border: 2px solid black;
   border-radius:10px;
```

```
}
button {
   position: absolute;
   top: 150px;
   width: 25px;
   height: 70px;
   font: 12pt arial,tahoma,sans-serif;
   text-align: center;
}
.left {
   left:5px;
}
.right {
   right:5px;
}
</style>
<script>
var slider = {
   slides:['6.jpg','9.jpg','20.jpg'],
   frame:0,
   set: function(image) {
      document.getElementById("scr").style.backgroundImage =
"url("+image+")";
   },
   init: function() {
      this.set(this.slides[this.frame]);
   },
   left: function() {
      this.frame--;
      if(this.frame < 0) this.frame = this.slides.length-1;
      this.set(this.slides[this.frame]);
   },
   right: function() {
      this.frame++;
      if(this.frame == this.slides.length) this.frame = 0;
      this.set(this.slides[this.frame]);
   }
};
window.onload = function() {
   slider.init();
   setInterval(function() {
      slider.right();
   },5000);
};
</script>
<body>
```

```
<div id="main">
   <button class="left" onclick="slider.left();"><</button>
   <div id="scr"></div>
   <button class="right" onclick="slider.right();">></button>
</div>
</body>
</html>
```

Как выглядит наш слайдер, показано на рис. 5.1.

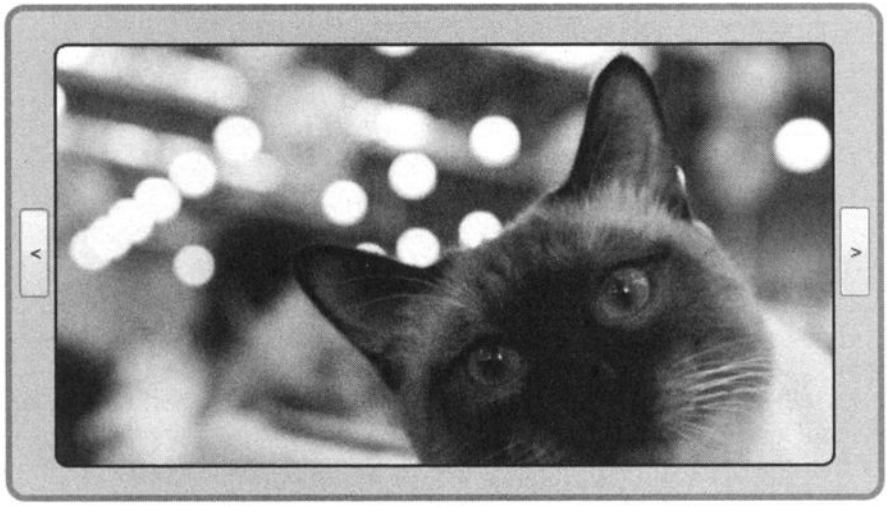


Рис. 5.1. Созданный вручную слайдер

5.2. Делаем слайдер средствами jQuery UI/Shoppica

Теперь посмотрим, как можно сделать слайдер средствами библиотеки jQuery. Первым делом нужно подключить необходимые сценарии и стили:

```
<script type="text/javascript" src="http://ajax.googleapis.
com/ajax/libs/jquery/1.5.2/jquery.min.js"></script>
<script type="text/javascript" src="http://ajax.googleapis.
com/ajax/libs/jqueryui/1.8.11/jquery-ui.min.js"></script>
<script type="text/javascript" src="js/shoppica.js"></script>
```

```
<link rel="stylesheet" type="text/css" href="stylesheet/960.
css" media="all" />
<link rel="stylesheet" type="text/css" href="stylesheet/
screen.css" media="screen" />
<link rel="stylesheet" type="text/css" href="stylesheet/color.
css" media="screen" />
```

Не волнуйтесь, все эти файлы можно будет скачать по предоставленному архиву. Заголовок слайдера будет выглядеть так:

```
<div id="intro">
<center><p><font color="#64a619" size=+2>Гироскутеры на любой
вкус</p></font></center>
    <div id="intro_wrap">
      <div id="product_intro" class="container_12">
        <div id="product_intro_info" class="grid_5">
```

Он будет постоянным в любом случае. Далее нужно описать сами продукты. Это делается так:

```
<div style="position: relative;">
    <h2><a href="product.php?id=244">Гироскутер IO CHIC Smart-
LS 9"</a></h2>
    <p class="s_desc">Модель IO CHIC Smart-LS весит всего лишь
21,5 кг, при этом она выдерживает максимальную нагрузку в 100
кг. Это удобный и компактный сигвей с двумя двигателями по 700
Вт и диаметром колес 9". Купить IO CHIC Smart-LS - это выгодно
и удобно.
</p>
     <div class="s_price_holder">
              <p class="s_price"> <span class="s_currency s_
before">1499</span> $ </p>
      </div>
</div>
```

Все, что вам остается, - это добавить описание своих продуктов в слайдер (добавить соответствующие div). Удобнее всего это делать PHP-сценарием, но поскольку книга не о нем, добавлять продукты будем вручную.

Далее нужно описать изображения для наших продуктов и кнопки назад/вперед:

```
<div id="product_intro_preview">
```

```
          <div class="slides_container"><div class="slideItem"
style="display: none;"><center>    <img
src="images/9.jpg" alt="Гироскутер IO CHIC Smart-LS 9 Black"
/></center></div>
<div class="slideItem" style="display:
none;"><center>    <img height=330 width=230
src="images/20" alt="Гироскутер IO CHIC Cross 20" /></
center></div>
<div class="slideItem" style="display:
none;"><center>    <img height=330 width=230
src="images/9.jpg" alt="IO CHIC Fairy 6.7 Red" /></center></
div>


            </div>
          <a class="s_button_prev" href="javascript:;"></a>
          <a class="s_button_next" href="javascript:;"></a>
```

И не забыть, ясное дело, подключить сценарии jQuery/Shoppica:

```
<script type="text/javascript" src="js/jquery/jquery.slides.
js"></script>
<script type="text/javascript" src="js/shoppica.products_
slide.js"></script>
```

Полный исходный код примера можно скачать (вместе со всеми необходимыми файлами - JavaScript и CSS) по адресу:

http://nit.com.ru/

В результате у нас должен получиться слайдер, изображенный на рис. 5.2.

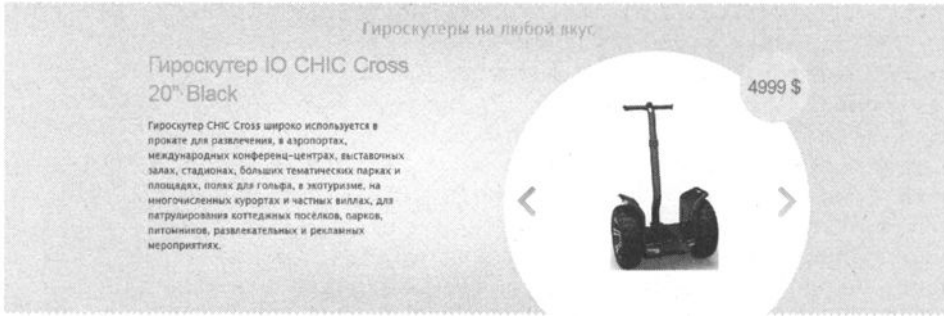
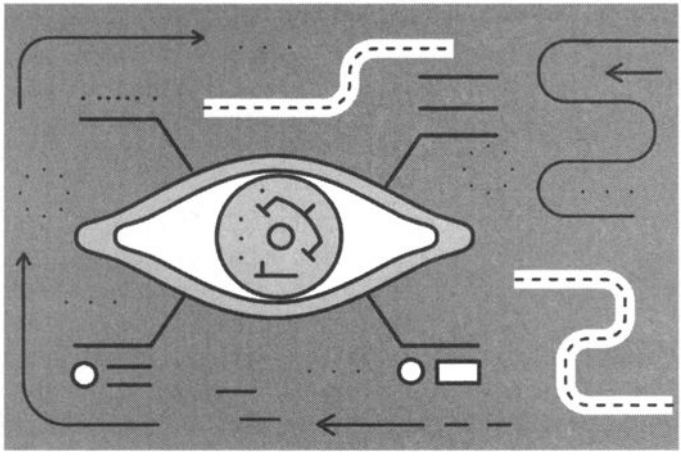
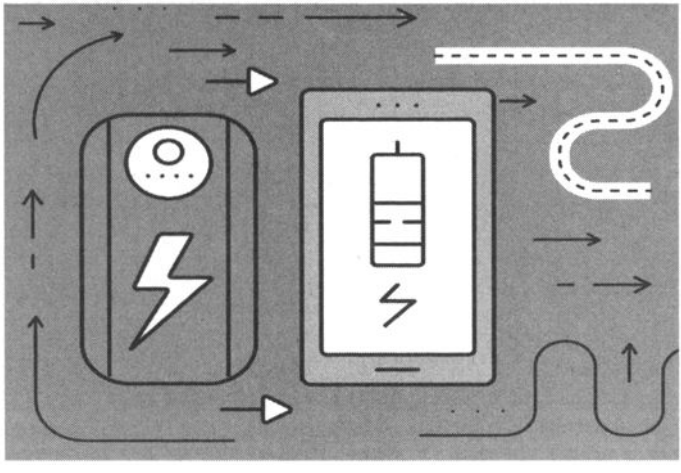


Рис. 5.2. Слайдер jQuery/Shoppica



Глава 6.

Красивые подсказки для сайта



Некоторые начинающие программисты пренебрежительно относятся к подсказкам. А зря. Подсказки удобно использовать как в панелях управления (чтобы пользователь точно знал, для чего используется та или иная кнопка), так и на основном сайте. Например, в нашем случае мы можем выводить подсказку с основными характеристиками гироскутеров в списке товаров. Это очень удобно - подвел мышку, увидел основные характеристики и уже потом открыл страничку товара. Так пользователю не придется открывать все товары подряд, чтобы ознакомиться с их характеристиками.

Всплывающие подсказки можно реализовать, как самостоятельно, так и использовать какую-нибудь библиотеку. Мы рассмотрим оба варианта - сначала сделаем красивые всплывающие подсказки сами, а потом будем использовать ToolTip.

6.1. Самостоятельное решение

Поскольку глава практическая, то сразу приступим к практике. Отройте ваш style.css и добавьте в него следующие строки (лист. 6.1).

Листинг 6.1. Стили для подсказки

```
/*всплывающие подсказки*/
#tt {position:absolute; display:block;
   padding:10px;
   border:solid #089dcb;
   border-width:1px;
   border-radius:10px 10px 10px 0px;
   -webkit-border-radius:10px 10px 10px 0px;
   -moz-border-radius:10px 10px 10px 0px;
   -webkit-box-shadow:1px 1px 2px #888888;
   -moz-box-shadow:1px 1px 2px #888888;
   box-shadow:1px 1px 2px #888888;
   background-color:#ffD;
}
.hlp1 { color:#000;border:#9fbddd 1px solid;background-
color:#E7F5FE;
   cursor:help;margin-left:5px;line-height:14px;
```

```
   width:12px;display:inline-block;text-align:center;
   border-radius:4px;
   -webkit-border-radius:4px;
   -moz-border-radius:4px;
   font-weight:100;
   text-indent:0;
}
```

При желании можете отредактировать цвет и параметры рамки. При наведении указателя мыши на объект сразу будет показана красивая всплывающая подсказка, использующая стили HTML5 для закругленных углов и теней. Если пользователь использует старый браузер, то подсказка тоже будет показана, только без всяких украшательств.

Теперь создайте файл tips.js (лист. 6.2) и поместите в него следующий код.

Листинг 6.2. Файл tips.js

```
var _tt=function(){
   var id = 'tt';
   var top = 3;
   var left = 3;
   var maxw = 300;
   var speed = 8;
   var timer = 10;
   var endalpha = 95;
   var alpha = 0;
   var tt,t,/*c,b,*/h;
   var ie = document.all ? true : false;
   return{
      show:function(e,v,w){
         var t=getEventTarget(e);addEvent(t,'mouseout',this.
hide); t.style.cursor='help';
         if(tt==null){
            tt=document.createElement('div');
            tt.setAttribute('id',id);
            document.body.appendChild(tt);
            tt.style.opacity=0;
            if(ie)tt.style.filter = 'alpha(opacity=0)';
         }
         tt.style.display = 'block';
         tt.innerHTML = v;
         tt.style.width = w ? w + 'px' : 'auto';
         if(tt.offsetWidth > maxw){tt.style.width=maxw+'px'}
         h = parseInt(tt.offsetHeight) + top;
         clearInterval(tt.timer);
```

```
        tt.timer=setInterval(function(){_tt.fade(1)},timer);
        dd=getOffset(t);
        tt.style.top = (dd.top-h+4) + "px";
        tt.style.left = (dd.left+13) + "px";

    },
    pos:function(e){
      var u = ie ? event.clientY + document.documentElement.
scrollTop : e.pageY;
      var l = ie ? event.clientX + document.documentElement.
scrollLeft : e.pageX;
      tt.style.top = (u - h) + 'px';
      tt.style.left = (l + left) + 'px';
    },
    fade:function(d){
      var a = alpha;
      if((a != endalpha && d == 1) || (a != 0 && d == -1)){
        var i = speed;
        if(endalpha - a < speed && d == 1){i = endalpha - a;
        }else if(alpha < speed && d == -1){i = a;}
        alpha = a + (i * d);
        tt.style.opacity = alpha * .01;
        if(ie)tt.style.filter='alpha(opacity=' + alpha + ')';
      }else{
        clearInterval(tt.timer);
        if(d == -1){tt.style.display = 'none'}
      }
    },
    hide:function(e){
      clearInterval(tt.timer);
      tt.timer = setInterval(function(){_tt.fade(-1)},timer);
    }
  };
}();

/* вспомогательная функция получения координат элемента */

function getOffset(elem) {
if(elem.getBoundingClientRect){
    var box = elem.getBoundingClientRect();
    var body = document.body;
    var docElem = document.documentElement;
    var scrollTop = window.pageYOffset || docElem.scrollTop ||
body.scrollTop;
    var scrollLeft = window.pageXOffset || docElem.scrollLeft
|| body.scrollLeft;
```

```
    var clientTop = docElem.clientTop || body.clientTop || 0;
    var clientLeft = docElem.clientLeft || body.clientLeft ||
0;
    var top  = box.top +  scrollTop - clientTop;
    var left = box.left + scrollLeft - clientLeft;
    return { top: Math.round(top), left: Math.round(left) }
}else{
    var top=0, left=0;
    while(elem) {
        top = top + parseInt(elem.offsetTop);
        left = left + parseInt(elem.offsetLeft);
        elem = elem.offsetParent;
    }
    return {top: top, left: left}
}
}

/* вспомогательная функция получения объекта, на котором
возникло событие */

function getEventTarget(e) {
  var e = e || window.event;
  var target=e.target || e.srcElement;
  if(typeof target == "undefined")return e; // передали this, а
не event
  if (target.nodeType==3) target=target.parentNode;// боремся
с Safari
  return target;
}

/* стандартная вспомогательная функция назначения обработчика
событий */

var addEvent = (function(){
   if (document.addEventListener){
           return function(obj, type, fn, useCapture){
                   obj.addEventListener(type, fn, useCapture);
   }
   } else if (document.attachEvent){ // для Internet Explorer
   return function(obj, type, fn, useCapture){
           obj.attachEvent("on"+type, fn);
   }
   } else {
   return function(obj, type, fn, useCapture){
           obj["on"+type] = fn;
   }
```

```
    }
})();
```

Кода очень много, поэтому мы вынесли его в отдельный файл. Однако нужно не забыть этот файл подключить, поэтому на каждой странице, где будут использоваться подсказки, нужно в секцию <head> добавить вот такой код:

```
<script src="tips.js"></script>
```

Осталось разобраться только, как использовать подсказки. Есть два варианта. Первый заключается в привязке подсказки к изображению товара, а именно:

```
  <img onmouseover="_tt.show(this,'Максимальная скорость - 10
км/ч&lt;br&gt;Максимальный вес - 90 кг.')" src="images/9.jpg"
alt="">
```

Здесь мы устанавливаем событие onmouseover для изображения. Обратите внимание, как осуществляется перенос строки: `<br>`

Второй вариант заключается в том, что мы выводим какой-то текст, например, *Инфо* или просто ? и привязываем к нему подсказку:

```
<span class="hlp1"
      onmouseover="_tt.show(this,'Максимальная скорость - 10
км/ч&lt;br&gt;Максимальный вес - 90 кг.')">?</span>
```

Именно для этого и предназначен второй класс hlp1. Давайте посмотрим, что вышло на практике (рис. 6.1). Слева на нем показано, что мы привязали подсказку к изображению. Она появится, как только пользователь подведет курсор мыши к картинке товара. Справа - что подсказка привязана к знаку ?. При подведении курсора мыши к этому знаку появится подсказка. Какой вариант использовать, решать только вам.

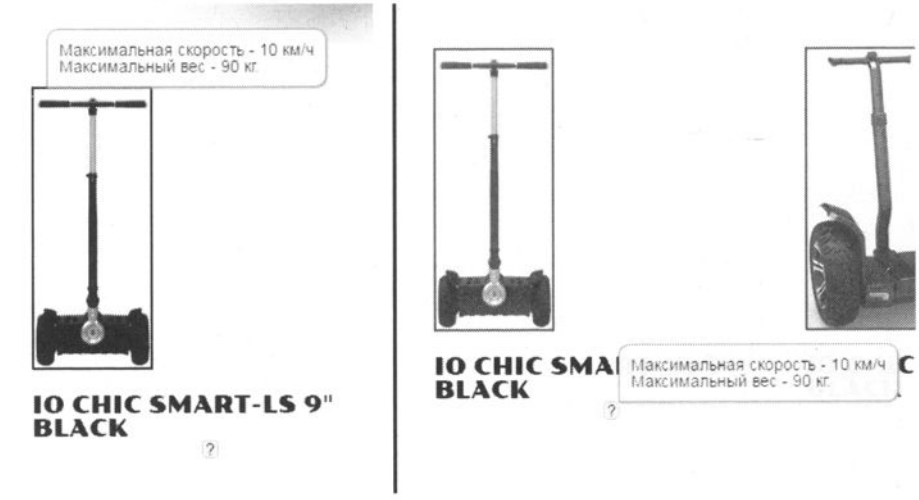


Рис. 6.1.

6.2. Скрипт ToolTip

Все сценарии подсказок работают одинаково - реагируют на событие OnMouseOver. По этому событию они должны отобразить подсказку. Внешний вид подсказки определяется только CSS и никак не зависит от самого скрипта. Он может просто использовать те или иные стили.

Вот как можно подключить готовый сценарий ToolTip:

```
<img src="images/20.jpg"
onmouseover="tooltip.show('Максимальная скорость - 20 км/
ч&lt;br&gt;Максимальный вес - 120 кг.');"
onmouseout="tooltip.hide();"
alt="">
```

Как видите, мы задаем не только событие OnMouseOver, но и OnMouseOut - это событие должно закрыть подсказку.

Стили могут выглядеть примерно так:

```
* {margin:0; padding:0}
body {font:11px/1.5 Verdana, Arial, Helvetica, sans-serif;
background:#FFF}
#text {margin:50px auto; width:500px}
```

```
.hotspot {color:#900; padding-bottom:1px; border-bottom:1px
dotted #900; cursor:pointer}

#tt {position:absolute; display:block; background:url(images/
tt_left.gif) top left no-repeat}
#tttop {display:block; height:5px; margin-left:5px;
background:url(images/tt_top.gif) top right no-repeat;
overflow:hidden}
#ttcont {display:block; padding:2px 12px 3px 7px; margin-
left:5px; background:#666; color:#FFF}
#ttbot {display:block; height:5px; margin-left:5px;
background:url(images/tt_bottom.gif) top right no-repeat;
overflow:hidden}
```

Как видите, стили компактнее, но и подсказка тоже будет выглядеть попроще предыдущего варианта.

Код JavaScript приведен в листинге 6.3.

Листинг 6.3. Файл tt.js

```
var tooltip=function(){
    var id = 'tt';
    var top = 3;
    var left = 3;
    var maxw = 300;
    var speed = 10;
    var timer = 20;
    var endalpha = 95;
    var alpha = 0;
    var tt,t,c,b,h;
    var ie = document.all ? true : false;
    return{
        show:function(v,w){
            if(tt == null){
                tt = document.createElement('div');
                tt.setAttribute('id',id);
                t = document.createElement('div');
                t.setAttribute('id',id + 'top');
                c = document.createElement('div');
                c.setAttribute('id',id + 'cont');
                b = document.createElement('div');
                b.setAttribute('id',id + 'bot');
                tt.appendChild(t);
                tt.appendChild(c);
                tt.appendChild(b);
```

```
                document.body.appendChild(tt);
                tt.style.opacity = 0;
                tt.style.filter = 'alpha(opacity=0)';
                document.onmousemove = this.pos;
            }
            tt.style.display = 'block';
            c.innerHTML = v;
            tt.style.width = w ? w + 'px' : 'auto';
            if(!w && ie){
                t.style.display = 'none';
                b.style.display = 'none';
                tt.style.width = tt.offsetWidth;
                t.style.display = 'block';
                b.style.display = 'block';
            }
            if(tt.offsetWidth > maxw){tt.style.width = maxw +
'px'}
            h = parseInt(tt.offsetHeight) + top;
            clearInterval(tt.timer);
            tt.timer = setInterval(function(){tooltip.
fade(1)},timer);
        },
        pos:function(e){
            var u = ie ? event.clientY + document.
documentElement.scrollTop : e.pageY;
            var l = ie ? event.clientX + document.
documentElement.scrollLeft : e.pageX;
            tt.style.top = (u - h) + 'px';
            tt.style.left = (l + left) + 'px';
        },
        fade:function(d){
            var a = alpha;
            if((a != endalpha && d == 1) || (a != 0 && d ==
-1)){
                var i = speed;
                if(endalpha - a < speed && d == 1){
                    i = endalpha - a;
                }else if(alpha < speed && d == -1){
                    i = a;
                }
                alpha = a + (i * d);
                tt.style.opacity = alpha * .01;
                tt.style.filter = 'alpha(opacity=' + alpha +
')';
            }else{
                clearInterval(tt.timer);
```

```
                if(d == -1){tt.style.display = 'none'}
            }
        },
        hide:function(){
            clearInterval(tt.timer);
            tt.timer = setInterval(function(){tooltip.fade(-
1)},timer);
        }
    };
}();
```

Чтобы его не перепечатывать, вы можете найти данный код в Интернете. Кстати, если вы сравните оба сценария, то обнаружите, что наша самостоятельная версия - это не что иное, как надстройка над исходным ToolTip. Если присмотритесь внимательнее, то обнаружите, что наш сценарий устанавливает другой тип курсора мыши при наведении на область с подсказкой, и много чего еще. Сравнение этих двух сценариев будет вашим домашним заданием. Внешний вид подсказки приведен на рис. 6.2.

Приведенные варианты создания всплывающих подсказок - не единственные. В Интернете вы найдете другие варианты. Но по мне смысла в этом нет, лучше поработать над оформлением подсказки (стилями).

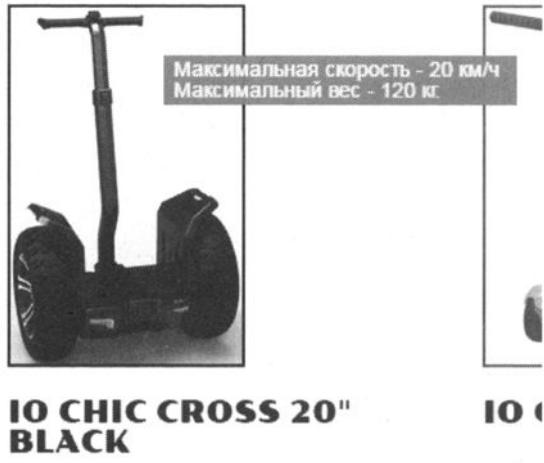
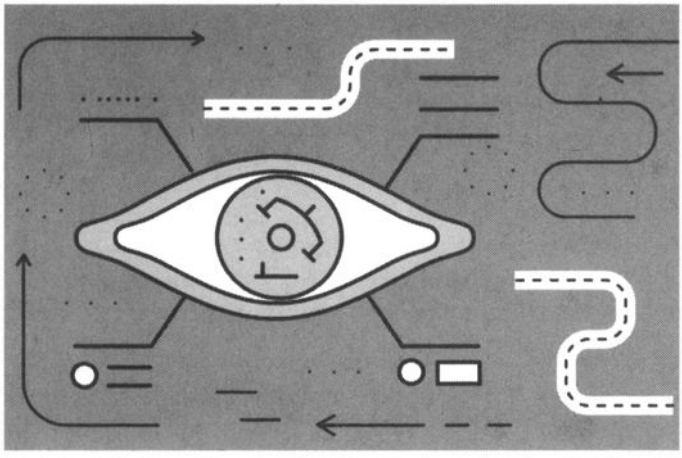
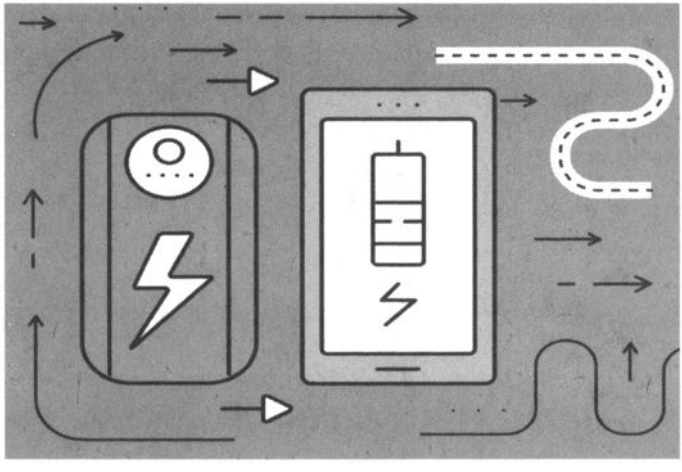


Рис. 6.2. Внешний вид подсказки (скрипт ToolTip)



Глава 7.

Функции



7.1. Основные понятия

Функция - это фрагмент JavaScript-кода, который можно вызвать из любого места основного сценария. По сути, функция - это подпрограмма. Функция описывается с помощью ключевого слова function, синтаксис описания функции следующий:

```
function <Имя функции> ([<Параметры>]) {
<Тело функции>
[return <Значение>]
}
```

Имя функции должно быть уникальным. Для него действуют те же правила, что и для имени переменной. После имени функции в круглых скобках указываются параметры функции. Если функции не передаются параметры, то указываются только круглые скобки. Если параметров несколько, то они разделяются запятыми.

В фигурных скобках располагаются выражения JavaScript. Как правило, они производят какие-то манипуляции над переданными параметрами.

Функция по определению должна возвращать результат. Результат возвращается с помощью ключевого слова return. Конечно, иногда нужно создать просто подпрограмму, которая не возвращает никакого результата (например, форматирует сообщение в диалоговом окне и выводит это самое диалоговое окно), в этом случае ключевое слово return не обязательно и можно обойтись без него.

Рассмотрим несколько примеров функций:

```
// Функция просто выводит диалоговое окно с текстом 'Access
denied'
// Использование этой функции просто короче, чем вызов
windows.alert
// К тому же, когда понадобится изменить текст сообщения,
тогда
// текст можно будет изменить в одном месте, а не по всему
сценарию
// Функция ничего не возвращает
```

```
function denied() {
window.alert('Access Denied!');
}

// Функция возвращает сумму двух чисел. Никакой проверки,
являются
// ли аргументы числами, не производится
function Sum(x, y) {
var result = x + y;
return result;
}
```

Использовать эти функции можно так:

```
denied();       // будет выведено наше сообщение
var x = Sum(2, 2);  // в переменную x будет записан результат
функции Sum
```

После инструкции return происходит выход из функции. Все операторы после оператора return не будут выполнены, например:

```
function Sum(x, y) {
var result = x + y;
return result;
window.alert('Сумма'); // никогда не будет выполнен

}
```

В конструкции return можно указать сразу выражение, перепишем нашу функцию Sum:

```
function Sum(x, y) {
return (x + y);
}
```

Функции можно передавать не только значения, но и значения переменных, например:

```
var a = 10;
var b = 12;

var ab = Sum(a, b);
```

Ссылку на функцию вы можете сохранить в любой переменной, например:

```
var d = denied;   // Имя функции указываем без скобок
d();      // Вызываем функцию denied() по ссылке
```

В JS допускаются также анонимные функции, то есть функции без названия:

```
var x = function() {            // ссылка на анонимную функцию
   window.alert('Тест');  // присваивается переменной x
}
x();            // вызываем функцию через переменную x
```

Ссылку на вложенную функцию можно возвратить в качестве значения конструкции return, для этого дважды используются круглые скобки. Пример:

```
var x = function() {            // Ссылка на анонимную функцию
return function() {       // Возвращаем ссылку на вложенную
функцию
window.alert("Тест");
};
   };
x()(); // Вызываем вложенную функцию
```

7.2. Расположение функций внутри сценария

Мы уже рассмотрели достаточно примеров функций, осталось только понять, где в HTML-документе должны находиться функции. Теоретически, функция может находиться в любом месте сценария, но до первого момента ее использования. Чтобы не запутывать прежде всего самих себя, программисты обычно помещают описание функций в секцию HEAD (заголовок) HTML-документа. Если же функций достаточно много или код функции слишком объемный, можно вынести код в отдельный .js файл. Сейчас мы рассмотрим оба варианта.

В листинге 7.1 я описал функцию в секции HEAD, а вызов функции производится из сценария секции BODY.

Листинг 7.1. Функция помещена в HEAD

```
<html>
<head>
  <title>Функции</title>
  <script>
    function denied() {
      window.alert('Access Denied!');
    }

  </script>
</head>
<body>
  <script>

    denied();     // вызываем функцию

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

В листинге 7.2 мы подключаем JS-файл functions.js (имя файла может быть любым). Код файла functions.js приведен в листинге 7.3.

Листинг 7.2. Вызов функции из внешнего JS-файла

```
<html>
<head>
  <title>Функции</title>
  <script type="text/javascript" src="functions.js"></script>
</head>
<body>
  <script>

  denied();

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

Листинг 7.3. Внешний JS-файл (functions.js)

```
function denied() {
   window.alert('Access Denied!');
}
```

Понятно, не нужно создавать отдельный JS-файл для каждой функции. Вы можете создать один-единственный файл, в который поместите все функции, необходимые вашему основному сценарию.

7.3. Рекурсия

Рекурсия – это явление, когда функция вызывает саму себя. Нужно отметить, что рекурсивные алгоритмы очень опасны и их рекомендуется по возможности избегать. Основная опасность в зацикливании, когда не предусмотрено (или предусмотрено некорректно) условие выхода из рекурсии. Во многих книгах по программированию рекурсия традиционно используется для вычисления факториала. Далее приведена функция Factorial(), вычисляющая факториал числа x. Условием выхода из рекурсии является оператор:

```
if (x == 0 || x == 1) return 1;
```

Если x равен 0 или 1, функция вернет 1, в противном случае она будет вычислять факториал x - 1, для чего вызовет саму себя.

Код функции:

```
function f_Factorial(x) {
if (x == 0 || x == 1) return 1;
else return (x * f_Factorial(x - 1));
}
```

7.4. Область видимости переменной: глобальные и локальные переменные

Глобальными являются все переменные, объявленные за пределами функции. Они доступны в любой части программы (сценария), в том числе и в функции.

Локальными являются переменные, объявленные в самой функции. Такие переменные доступны только функции, в которой они объявлены, и недоступны в других функциях или в основной программе.

Если имя локальной переменной совпадает с именем глобальной переменной, то будет использоваться локальная переменная, а значение глобальной переменной останется без изменения.

Рассмотрим листинг 7.4. В секции HEAD мы объявляем две переменные - A и B. Они будут глобальными переменными, доступными как в функции F1(), так и в коде из секции BODY, то есть везде по сценарию. В теле функции мы объявляем две локальные переменные - X и B. Затем функция выводит значение переменных A, B, X. Посмотрите на рис. 7.1. Функция вывела значения 10, 5 и 10. Как видите, используется локальная переменная B вместо глобальной переменной B, если имена переменных совпадают. Основная программа выводит значения переменных A и B, будут выведены значения 10 и 20. Выводить значение переменной X в основной программе нет смысла, так как вы получите сообщение об ошибке **Uncaught ReferenceError: X is not defined**.

Листинг 7.4. Глобальные и локальные переменные

```
<html>
<head>
  <title>Глобальные и локальные переменные</title>
  <script>
  // глобальные переменные
  var A = 10;
  var B = 20;

  function F1() {

  // локальные переменные
  var X = 10;
  var B = 5;

  document.write("<br>A = " + A);
  document.write("<br>B = " + B);
  document.write("<br>X = " + X);

  }

  </script>
</head>
<body>
  <script>

  F1();

  document.write("<HR>");
  document.write("A = " + A);
  document.write("<br>B = " + B);
```

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

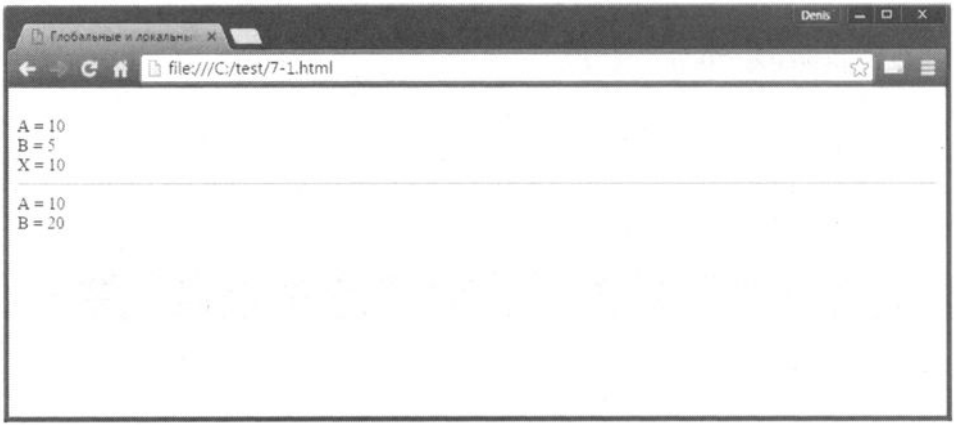
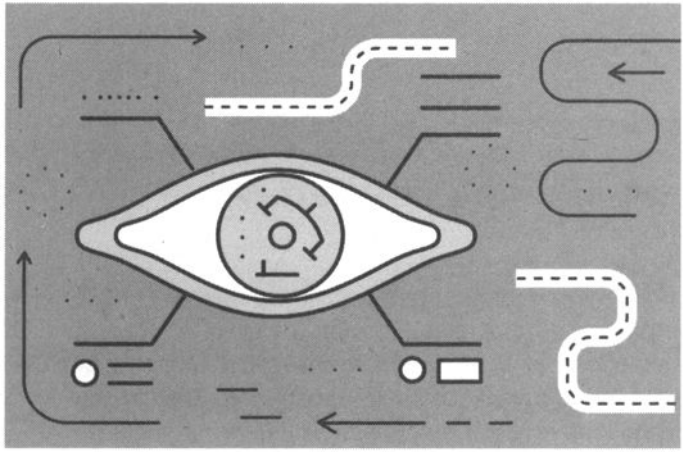
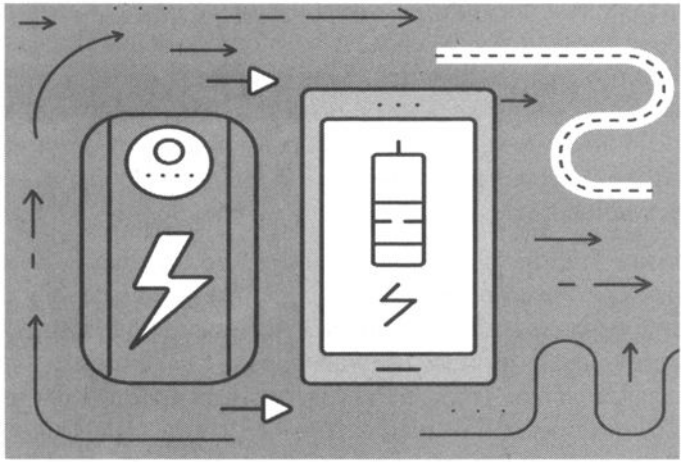


Рис. 7.1. Область видимости локальных и глобальных переменных



Глава 8.
Основы объектно-ориентированного программирования



8.1. Основные концепции

Объектно-ориентированное программирование (ООП) — это особый подход к написанию программ. Чтобы понять, что такое ООП и зачем оно нужно, необходимо вспомнить некоторые факты из истории развития вычислительной техники. Первые программы вносились в компьютер с помощью переключателей на передней панели компьютера - в то время компьютеры занимали целые комнаты. Такой способ "написания" программы, сами понимаете, был не очень эффективным - ведь большая часть времени (несколько часов, иногда - целый рабочий день) занимало подключение кабелей и установка переключателей. А сами расчеты занимали считанные минуты. Вы только представьте, что делать, если один из программистов (такие компьютеры программировались, как правило, группами программистов) неправильно подключил кабель или установил переключатель? Да, приходилось все перепроверять - по сути, все начинать заново.

Позже появились перфокарты. Программа, то есть последовательность действий, которые должен был выполнить компьютер, наносилась на перфокарту. Пользователь вычислительной машины (так правильно было называть компьютеры в то время) писал программу, оператор "записывал" программу на перфокарту, которая передавалась оператору вычислительного отдела. Через определенное время оператор возвращал пользователю результат работы программы - рулон бумаги с результатами вычислений. Мониторов тогда не было, а все, что выводил компьютер, печаталось на бумаге. Понятно, если в расчетах была допущена ошибка (со стороны пользователя, компьютеры ведь не ошибаются - они делают с точностью то, что заложено программой), то вся цепочка действий (программист, оператор перфокарты, оператор вычислительной машины, проверка результатов) повторялась заново.

Следующий этап в программировании - это появление языка Ассемблер. Этот язык программирования позволял писать довольно длинные для того времени программы. Но Ассемблер - это язык программирования низкого уровня, все операции проводятся на уровне "железа". Если вы не знаете, то сейчас я вам поясню. Чтобы в PHP выполнить простейшее действие, например сложение, достаточно записать '$A = 2 + 2; '. На языке Ассемблер вам

для выполнения этого же действия нужно было выполнить как минимум три действия - загрузить в один из регистров первое число (команда MOV), загрузить в другой регистр второе число (опять команда MOV), выполнить сложение регистров командой ADD. Результат сложения будет помещен в третий регистр. Названия регистров я специально не указывал, поскольку они зависят от архитектуры процессора, а это еще один недостаток Ассемблера. Если вам нужно перенести программу на компьютер с другой архитектурой, вам нужно переписать программу с учетом особенностей целевой архитектуры.

Требования к программным продуктам и к срокам их разработки росли (чем быстрее будет написана программа, тем лучше), поэтому появились языки программирования высокого уровня. Язык высокого уровня позволяет писать программы, не задумываясь об архитектуре вашего процессора. Нет, это не означает, что на любом языке высокого уровня можно написать программу, которая в итоге станет работать на процессоре с любой архитектурой. Просто при написании программы знать архитектуру процессора совсем не обязательно. Вы пишете просто A = B + C и не задумываетесь, в каком из регистров (или в какой ячейке оперативной памяти) сейчас хранятся значения, присвоенные переменным B и C. Вы также не задумываетесь, куда будет помещено значение переменной A. Вы просто знаете, что к нему можно обратиться по имени A. Первым языком высокого уровня стал FORTRAN (FORmula TRANslator).

Следующий шаг - это появление структурного программирования. Дело в том, что программы на языке высокого уровня очень быстро стали расти в размерах, что сделало их нечитабельными из-за отсутствия какой-либо четкой структуры самой программы. Структурное программирование подразумевает наличие структуры программы и программных блоков, а также отказ от инструкций безусловного перехода (GOTO, JMP).

После выделения структуры программы появилась необходимость в создании подпрограмм, которые существенно сокращали код программы. Намного проще один раз написать код вычисления какой-то формулы и оформить его в виде процедуры (функции) - затем для вычисления 10 результатов по этой формуле нужно будет 10 раз вызвать процедуру, а не повторять 10 раз один и тот же код. Новый класс программирования стал называться процедурным.

Со временем процедурное программирование постигла та же участь, что и структурное, - программы стали настолько большими, что их было неудобно читать. Нужен был новый подход к программированию. Таким стало объектно-ориентированное программирование (далее - ООП).

ООП базируется на трех основных принципах - инкапсуляция, полиморфизм, наследование. Разберемся, что есть что.

С помощью инкапсуляции вы можете объединить воедино данные и обрабатывающий их код. Инкапсуляция защищает и код, и данные от вмешательства извне. Базовым понятием в ООП является класс. Грубо говоря, класс - это своеобразный тип переменной. Экземпляр класса (переменная типа класс) называется объектом. В свою очередь, объект - это совокупность данных (свойств) и функций (методов) для их обработки. Данные и методы обработки называются членами класса.

Получается, что объект - это результат инкапсуляции, поскольку он включает в себя и данные, и код их обработки. Чуть дальше вы поймете, как это работает, пока представьте, что объект - это эдакий рюкзак, собранный по принципу "все свое ношу с собой".

Члены класса могут быть открытыми или закрытыми. Открытые члены класса доступны для других частей программы, которые не являются частью объекта. Закрытые члены доступны только методам самого объекта.

Теперь поговорим о полиморфизме. Если вы программировали на языке C (на обычном C, не C++), то наверняка знакомы с функциями abs(), fabs(), labs(). Все они вычисляют абсолютное значение числа, но каждая из функций используется для своего типа данных. Если бы C поддерживал полиморфизм, то можно было бы создать одну функцию abs(), но объявить ее трижды - для каждого типа данных, а компилятор бы уже сам выбирал нужный вариант функции, в зависимости от переданного ей типа данных. Данная практика называется перезагрузкой функций. Перезагрузка функций существенно облегчает труд программиста - вам нужно помнить в несколько раз меньше названий функций для написания программы.

Полиморфизм позволяет нам манипулировать с объектами путем создания стандартного интерфейса для схожих действий.

Осталось поговорить о наследовании. С помощью наследования один объект может приобретать свойства другого объекта. Заметьте, наследование - это не копирование объекта. При копировании создается точная копия объекта, а при наследовании эта копия дополняется уникальными свойствами (новыми членами). Наследование можно сравнить с рождением ребенка, когда новый человек наследует "свойства" своих родителей, но в то же время не является точной копией одного из родителей.

Все вышесказанное было истинно для любого полноценного объектно-ориентированного языка программирования. В JavaScript поддержка ООП довольно ограничена. Например, нет ни закрытых (приватных), ни открытых

свойств и методов. Все свойства и методы являются открытыми. Создание класса сводится к созданию функции конструктора (будет показано ниже). Полноценной поддержки ООП в JavaScript нет и не будет - это вам не Java, а просто скриптинговый язык для браузера. Далее мы рассмотрим объектно-ориентированные возможности JavaScript.

8.2. Создание пользовательских классов и объектов

Мы уже знакомы немного с классами и объектами. Создать новый объект можно с помощью встроенного класса Object, например:

```
var Human = new Object();
// свойства объекта
Human.firstname = "John";
Human.lastname = "Doe";
// формируем метод объекта
Human.getFullName = function() {
   var fname = this.firstname + this.lastname;
   return fname;
}

// просто выводим значения:
window.alert(Human.firstname);
window.alert(Human.lastname);
window.alert(Human.getFullName);
```

После того как объект создан, на него в переменной Human сохраняется ссылка. В качестве значения свойства объекта можно использовать любой тип данных - число, массив, строку или другой объект. Если в качестве значения указана ссылка на функцию, то такое свойство становится методом объекта, внутри которого доступен указатель на текущий объект (this).

Если вы привыкли к другим языкам программирования и такое создание объекта вам непривычно, можете использовать фигурные скобки для определения свойств и методов объекта:

```
var Human = {
   firstname: "John";
   lastname: "Doe";
   getFullName: function() {
        var fname = this.firstname + this.lastname;
```

```
        return fname;
    }
}
```

В этом случае значение свойств/методов указывается через двоеточие. Если в фигурных скобках нет никаких выражений, тогда создается пустой объект:

```
var empty_obj = {};
```

Есть один очень важный момент, который вы должны осознать. Пусть нам нужно создать два одинаковых объекта, которые должны использоваться раздельно. Вот что первым приходит в голову:

```
var o1 = o2 = {};
```

Однако так нельзя делать! Ведь создается только один объект, а ссылка на него сохраняется в двух переменных. Поэтому все изменения в переменной o1 будут отображаться и на переменной o2 и наоборот:

```
o1.firstname = "John";
document.write(o2.firstname);    // будет выведено John
```

Что же делать? Нужны два разных оператора инициализации переменной-объекта, например:

```
var o1 = {};
var o2 = {};
```

Если после ключевого слова new указывается функция, то она становится конструктором объекта. Такой функции можно передать начальные данные для инициализации объекта (можно ничего не передавать, все зависит от реализации). Функции-конструкторы удобно использовать, если нужно инициализировать несколько подобных объектов. Рассмотрим функцию-конструктор:

```
function Human(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
    this.getFullName = function() {
            var fname = this.firstname + this.lastname;
        return fname;
     }
```

```
}

var John = new Human("John", "Doe");
var Ivan = new Human("Ivan", "Ivanov");
```

Что такое класс? Класс - это тип объекта, включающий в себя переменные и функции для управления этими переменными. Да, переменные - это свойства, а функции - это методы. Создать экземпляр класса, то есть объект, можно так:

```
<экземпляр> = new <имя объекта> ([параметры]);
```

Теперь посмотрите на то, как мы создали объекты John и Ivan. Мы указали служебное слово new, затем имя нашей функции и передали ей параметры. Другими словами, создав функцию-конструктор, мы создали класс Human. Вот так вот. Как я уже и говорил, поддержка ООП в JavaScript несколько изощренная. Если вы никогда не программировали на объектно-ориентированном языке программирования, то ничего не обычного не заметите. Но если же вы программировали хотя бы на том же PHP, то создание класса посредством создания функции-конструктора - это дикость. Но есть то, что есть, и вам с этим придется смириться и использовать именно в таком контексте, как это предлагает JavaScript.

Зато в JavaScript есть удобный цикл for..in, позволяющий пройтись по всем свойствам объекта. Это дает возможность вывести свойства объекта, которые ранее были неизвестны. А так как у объектов нет закрытых свойств, то будут выведены абсолютно все свойства объекта. Пример:

```
for (var P in John) {
document.write(P + " = " + John[P]);
}
```

Обратите внимание: мы обращаемся к объекту, как к массиву (в PHP такие массивы называются ассоциативными, где в качестве индекса может использоваться не только число, но и строка).

Оператор in позволяет проверить существование свойства в объекте, например:

```
if ("firstname" in John) window.alert(John.firstname);
```

Проверить наличие метода можно, указав его имя без скобок:

```
if (John.getFullName) window.alert('getFullName exists');
```

Так нельзя проверять наличие свойства, поскольку значение 0 будет интерпретироваться как false, в итоге такой проверки вы получите, что свойство не существует, но на самом деле оно наличествует, но просто равно 0.

С помощью оператора instanceof можно проверить принадлежность экземпляра классу, например:

```
if ((typeof John == "object") && (John instanceof Human))
   window.alert('John is instance of Human');
```

Удалить свойство можно так:

```
delete Ivan.lastname;
```

8.3. Прототипы

Ранее мы определяли метод getFullName() внутри конструктора:

```
function Human(firstname, lastname) {
   this.firstname = firstname;
   this.lastname = lastname;
   this.getFullName = function() {
          var fname = this.firstname + this.lastname;
       return fname;
    }
}
```

Такое решение не всегда эффективно. Например, нужно создать массив, состоящий из 1000 объектов. При этом свойства будут разные для всех объектов, а метод getFullName() одинаков для всех.

Использование прототипов позволяет определить метод вне конструктора. При создании объекта будут унаследованы все свойства, которые имеются в прототипе. Поэтому метод getFullName будет определен всего один раз, но унаследуется всеми экземплярами класса.

Рассмотрим пример:

```
function Human(firstname, lastname) {
   this.firstname = firstname;
   this.lastname = lastname;
}

Human.prototype.getFullName = function() {
```

```
var fname = this.firstname + this.lastname;
     return fname;
}

var John = new Human("Анна", "Машкина");
document.write(John.getFullName());          // Анна Машкина

var Ivan = new Human("Федор", "Иванов");
document.write(Iva.getFullName());           // Федор Иванов
```

8.4. Пространства имен

Представим, что вы написали функцию DecodeString(), а затем подключили какую-то библиотеку, в которой есть функция с таким же названием. Произойдет конфликт имен. В результате будет использована та функция, которая была определена последней. Ясно, что параметры этих двух функций могут отличаться, и если вы попытаетесь вызвать свою функцию, указав 3 параметра, а последней была определена функция DecodeString() с двумя параметрами, выполнение сценария будет остановлено с ошибкой.

Чтобы этого не произошло, используются пространства имен. Каждая функция будет определена в своем пространстве имен, а вам нужно будет указывать, из какого пространства имен та или иная функция вызывается.

В JavaScript в качестве пространств имен используются объекты. Созданный экземпляр объекта помещается в глобальную область видимости. а все остальные идентификаторы будут доступны через свойства объекта, например:

```
var MyLibrary = {};
MyLibrary.DecodeString = function() {
   document.write('Тест');
}

MyLibrary.DecodeString();
```

В данном примере функция DecodeString() определена внутри пространства имен MyLibrary. Понятно, что конфликт имен сводится к минимуму (конечно, если вы не вздумали назвать свой объект так, как называется сторонняя библиотека).

Если вы занимаетесь JavaScript-разработкой профессионально и пишете скрипты для разных сайтов, рекомендуется создавать пространства имен, совпадающие с именем сайта. Так вы полностью исключите возможность

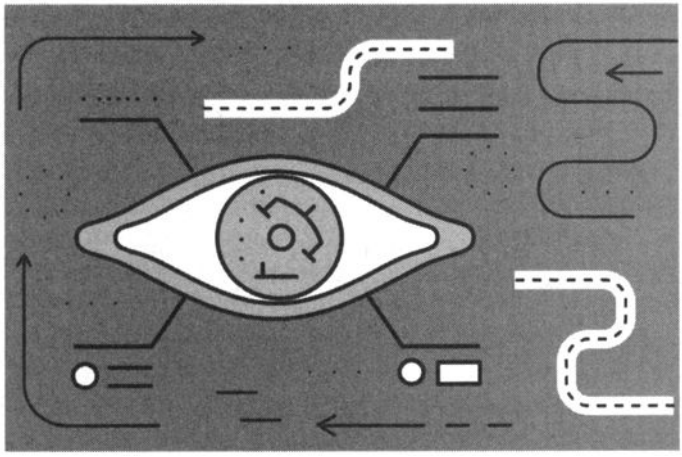
конфликта имен, а чтобы было удобнее обращаться к свойствам и методам, используйте доступ по ссылке. Например:

```
var dkwsorgua = {};          // создаем пространство имен
var $ = dkwsorgua;           // создаем ссылку на пространство
имен
```

Далее используем ссылку $ (чтобы не переписывать весь код, когда вам нужно реализовать подобные функции на разных сайтах), например:

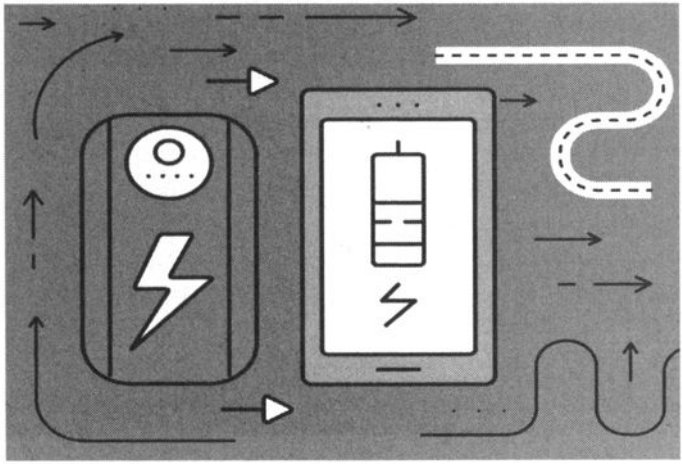
```
$.DecodeString();
```

Здесь мы разобрались, как создавать собственные классы и объекты в JavaScript. В главе 12 рассмотрим встроенные классы JavaScript. А пока - немного практики, ведь нам нужно не только изучать JavaScript, но и совершенствовать наш сайт.



Глава 9.

Меню и панели



Меню - очень важный элемент любого сайта, и от того, как оно устроено (и как продумано), во многом зависит, найдет ли пользователь необходимый материал на сайте и как комфортно ему будет работать с вашим сайтом. Конечно, необходимый материал можно найти через поисковик, но в этом случае пользователь увидит много сайтов ваших конкурентов, и если один из них будет более удобным в использовании, чем ваш, не думаю, что у пользователя возникнет желание вернуться на ваш сайт.

Статические меню уже давно канули в прошлое. Они неудобны, и при выборе пункта меню, чтобы появились дочерние элементы меню, приходится ждать перезагрузки страницы. Во-первых, это время ожидания для пользователя. А если соединение медленное или сервер слишком загружен, ждать приходится несколько секунд, что очень раздражает. Во-вторых, как ни крути, каждый такой запрос - лишняя нагрузка на сервер. Допустим, на сайт зашло 1000 пользователей и выбрало один из пунктов меню, чтобы увидеть команды меню, в итоге была создана лишняя тысяча запросов, которых можно было бы избежать.

9.1. Делаем меню вручную

Для ценителей чистого JavaScript (без использования сторонних библиотек) первым делом рассмотрим простое меню, а в следующем разделе создадим динамическое меню средствами SuperFish. Однако оно будет динамическим, то есть раскрываться и отображать подпункты меню.

Создать динамическое меню, оказывается, очень просто. Принцип следующий. В HTML-коде мы формируем блоки. Каждый блок будет представлять собой отдельный пункт меню или подменю. Блоки мы будем формировать с помощью таблиц. У нас будет одна таблица для пунктов главного меню, а также отдельные таблицы для каждого подменю. Строки таблицы будут пунктами данного подменю.

При наведении указателя мыши на пункт главного меню нужно отобразить заданное подменю, например:

```
onMouseOver="getElementById('menu1').style.display='block';"
```

Когда указатель мыши уходит за пределы этого пункта меню, подменю нужно скрыть:

```
onMouseOut="getElementById('menu1').style.display='none';"
```

Подменю menu1 описывается так:

```
<table class="ids" id="menu1" onMouseOut="this.style.
display='none';"
onMouseOver="this.style.display='block';">
 <tr><td><a href="#">Мой профиль</td></tr>
 <tr><td><a href="#">Мои заказы</td></tr>
 <tr><td><a href="#">Корзина</td></tr>
</table>
```

Принцип прост, поэтому сразу привожу полный код примера (лист. 9.1).

Листинг 9.1. Динамическое меню (чистый JS)

```
<html>
<head>
<style type="text/css">
.menu {
    position:relative;
    height:30px;
}
.menu table {
    position: absolute;
    left: 10px;
    top: 0px;
    border: 1px solid black;
    background-color: white;
    border-spacing: 0;
}
.menu td {
    width: 200px;
    padding: 5px;
    text-align: center;
}
.menu a{
    text-decoration: none;
    font: 12px Verdana, Arial, sans-serif;
    color: blue;
    display: block;
```

```
    border: 1px solid black;
}
.menu a:hover{
    text-decoration: underline;
    color: black;
}
.ids td {
    width: 188px;
}
.ids a {
    border: none;
}
#menu1, #menu2, #menu3, #menu4 {
    display:none;
    top: 20px;
    left: 16px;
}
#menu2 {
    left: 226px;
}
#menu3 {
    left: 436px;
}
#menu4 {
    left: 646px;
}
</style>
</head>
<body>
<!-- меню первого уровня -->
<div class="menu">
<table>
 <tr>
  <td>
   <a href="#" onMouseOut="getElementById('menu1').style.
display='none';"
   onMouseOver="getElementById('menu1').style.
display='block';">Главная</a>
  </td>
  <td>
   <a href="#" onMouseOut="getElementById('menu2').style.
display='none';"
   onMouseOver="getElementById('menu2').style.
display='block';">Доставка и оплата</a>
  </td>
   <td>
```

```
   <a href="#" onMouseOut="getElementById('menu3').style.
display='none';"
   onMouseOver="getElementById('menu3').style.
display='block';">Каталог</a>
  </td>
  <td>
   <a href="#" onMouseOut="getElementById('menu4').style.
display='none';"
   onMouseOver="getElementById('menu4').style.
display='block';">Контакты</a>
  </td>
 </tr>
</table>
<!-- подменю -->
<table class="ids" id="menu1" onMouseOut="this.style.
display='none';"
onMouseOver="this.style.display='block';">
 <tr><td><a href="#">Мой профиль</td></tr>
 <tr><td><a href="#">Мои заказы</td></tr>
 <tr><td><a href="#">Корзина</td></tr>
</table>
<table class="ids" id="menu2" onMouseOut="this.style.
display='none';"
onMouseOver="this.style.display='block';">
 <tr><td><a href="#">Способы оплаты</td></tr>
 <tr><td><a href="#">Способы доставки</td></tr>
 <tr><td><a href="#">Гарантии</td></tr>
</table>
<table class="ids" id="menu3" onMouseOut="this.style.
display='none';"
onMouseOver="this.style.display='block';">
 <tr><td><a href="#">Взрослым</td></tr>
 <tr><td><a href="#">Детям</td></tr>
</table>
<table class="ids" id="menu4" onMouseOut="this.style.
display='none';"
onMouseOver="this.style.display='block';">
 <tr><td><a href="#">Адрес и карта проезда</td></tr>
 <tr><td><a href="#">Обратная связь</td></tr>

</table>
</div>
```

К преимуществам этого меню можно отнести простоту реализации и независимость от сторонних библиотек. Внешний вид (рис. 9.1) можно настроить с помощью CSS. Но у нашего решения есть и недостатки. Динамическая

библиотека сама вычисляет размеры подменю и сама определяет начальное положение подменю. В нашем случае начальная позиция жестко прописывается в CSS и никак не привязана к размеру экрана. Все это, конечно, накладывает определенные ограничения на использование нашего меню. Вы можете усовершенствовать его, а можете не изобретать велосипед и использовать Superfish для построения меню.

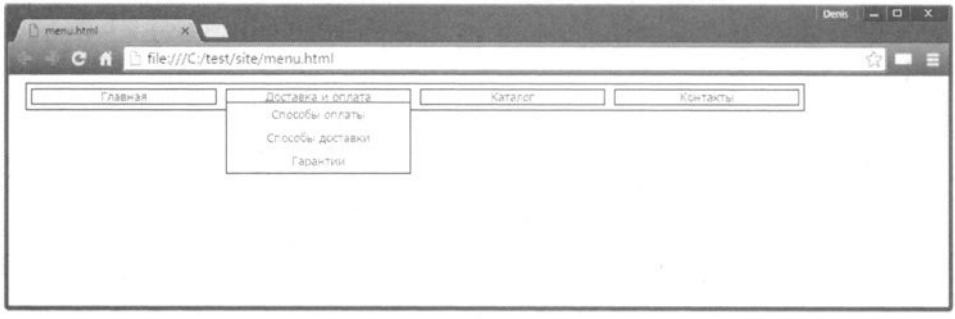


Рис. 9.1. Внешний вид

9.2. Динамическое меню средствами Superfish

9.2.1. Создание меню

Ранее написание динамического меню на JavaScript было неблагодарной работой. Приходилось писать его с нуля, а это задача не из легких. Конечно, кто-то уже написал свое динамическое меню, но после того, как просмотришь с десяток таких вариантов, приходишь к выводу, что проще написать самому. Также помню программы, генерирующие динамические меню: вы задаете структуру меню в программе, а она генерирует код, который нужно было вставить в вашу страницу. Плохо только, что внешний вид такого меню не поддавался изменению (или же за дополнительные деньги приходилось покупать программу-генератор меню), а что-либо изменить в запутанном коде было сложно.

С появлением jQuery все стало гораздо проще. Теперь вы за считанные минуты разработаете свое меню. Правда, это заслуга не одной только jQuery. Мы будем использовать дополнительный плагин Superfish (https://github.com/joeldbirch/superfish/) для jQuery. Этот плагин позволяет создавать очень неплохие меню. Также нам понадобится плагин hoverIntent, который

позволяет заменить обработчик hover (позже вы поймете, для чего это нужно). Теоретически, если вы не будете работать с обработчиками hover и вам не нужен плагин hoverIntent, его можно не использовать, чтобы не перегружать программу лишним кодом. Основной функционал меню будет доступен и без этого плагина.

Секция <head> нашего документа со всеми необходимыми подключенными сценариями будет выглядеть так:

```
<head>
  <title>Меню</title>
  <link rel="stylesheet" type="text/css" href="css/superfish.
css" />
  <script type="text/javascript" src="jquery-1.x.x.min.js"></
script>
  <script type="text/javascript" src="hoverIntent.js"></
script>
  <script type="text/javascript" src="superfish.js"></script>
 ...
```

Мы подключили CSS-файл superfish.css, который находится в подкаталоге css, библиотеку jQuery, а также плагины hoverIntent и Superfish.

Благодаря плагину Superfish создать меню очень и очень просто. Весь JavaScript-код будет следующим:

```
<script type="text/javascript">
$(function(){
  $("ul.sf-menu").superfish();
});
</script>
```

Само меню определяется в HTML-коде в виде списка. Нужно определить ненумерованный список (<ul>) стиля "sf-menu" (этот стиль используется, чтобы все списки на странице не превратились в меню). Далее, каждый элемент списка (<li>) превращается в меню. Если вам нужно определить подэлементы меню, внутри <li> вы определяете еще один список <ul>. Вложенность списков не ограничивается, поэтому внутри <ul> могут быть <li>, внутри которых определены еще списки <ul>. Однако не следует увлекаться иерархией списков, поскольку вы можете сделать меню очень запутанным. Оптимальным считается меню, имеющее 2-3 уровня иерархии.

Далее приведено описание меню для нашего сайта гироскутеров (лист. 9.2).

Листинг 9.2. Меню для сайта гироскутеров

```
<ul class="sf-menu">
  <li class="current">
    <a href="index.html">Главная</a>
    <ul class="A">
      <li><a href="index.html">Мой профиль</a></li>
      <li class="current">
        <a href="#ab">Мои заказы</a>
        <ul class="AA">
          <li class="current"><a href="index.html">Корзина</
a></li>
          <li><a href="index.html">Оформить заказ</a></li>
          <li><a href="index.html">Очистить все</a></li>
          <li><a href="index.html">Просмотреть предыдущие
заказы</a></li>
        </ul>
      </li>
      </li>
    </ul>
  </li>

  <li>
        <a href="#">Доставка и оплата</a>
        <ul class="AB">
          <li><a href="index.html">Как оплатить</a></li>
          <li><a href="index.html">Способы доставки</a></li>
          <li><a href="index.html">Гарантия</a></li>
         </ul>
      </li>

  <li>
    <a href="#">Каталог</a>
    <ul class="B">
      <li>
        <a href="#">Взрослым</a>
        <ul class="BA">
          <li><a href="index.html">До 10 км/ч</a></li>
          <li><a href="index.html">10-15 км/ч</a></li>
          <li><a href="index.html">Свыше 15 км/ч</a></li>
        </ul>
      </li>
      <li>
        <a href="#">Детям</a>
        <ul class="BB">
          <li><a href="index.html">до 5 км/ч</a></li>
```

```
            <li><a href="index.html">5-10 км/ч</a></li>

          </ul>
        </li>

      </ul>
    </li>
    <li><a href="index.html">Контакты</a></li>
</ul>
```

Обратите внимание: для некоторых пунктов меню определены подменю второго и третьего уровней, а пункт Контакты остался у нас без подменю. Такое тоже возможно - не все пункты меню требуют подменю.

Не забудьте закрыть главный список ! Вообще самая большая сложность заключается в необходимости следить за иерархией списков и не забывать закрывать их. Нужно быть очень внимательным при описании меню, иначе можете получить непредсказуемые результаты, вплоть до того, что обычные списки на вашей странице будут преобразованы в меню.

Результат приведен на рис. 9.2.

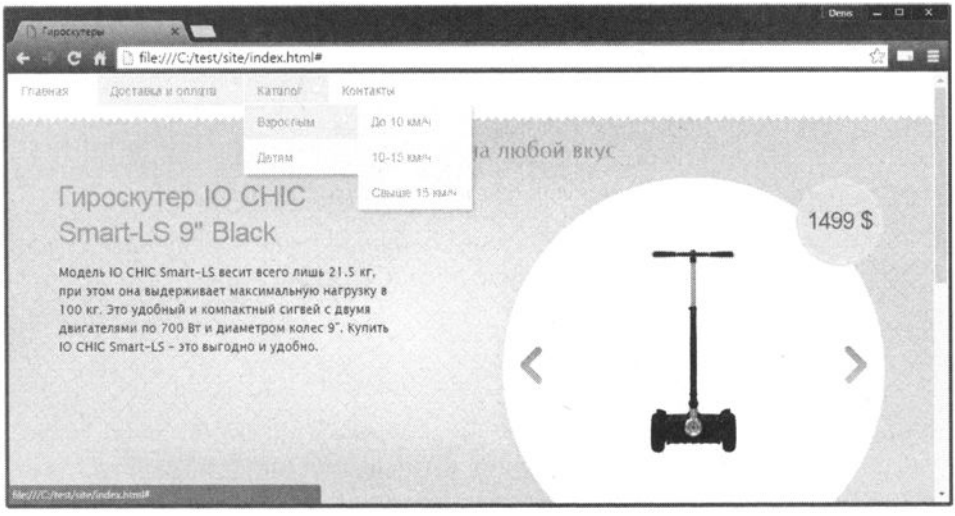


Рис. 9.2. Динамическое меню средствами Superfish

Когда вы будете создавать меню для собственного сайта (а не для сайта-примера), его внешний вид необходимо адаптировать под дизайн вашего сайта. Для этого вам нужно или отредактировать файл superfish.css вручную или же скачать одну из тем оформления для Superfish с сайта http://www.sooperthemes.com/category/tags/superfish. Темы оформления на этом сайте платные, но достаточно красивые и подойдут для коммерческих проектов. Конечно, можно также найти бесплатные темы в Интернете.

9.2.2. Настройка меню

Как видите, создать меню с помощью Superfish достаточно просто. Попытаемся теперь его настроить. Начнем с того, что мы добавим анимацию. Измените код, отображающий меню, так:

```
$(function(){
  $("ul.sf-menu").superfish({
    animation: { height: "show" },
    delay: 1500
  });
});
</script>
```

Теперь меню будет отображаться плавно и с анимационным эффектом. Параметр delay задает задержку в 1.5 секунды: именно столько меню будет еще отображаться после выхода указателя мыши за его пределы.

Если при наведении указателя мыши на пункт меню нужно выполнять какое-то действие, задайте обработчик для события onShow:

```
onShow: function(){ alert("test");
```

При подведении мыши к каждому пункту меню будет отображено диалоговое окно со строкой "test". Конечно, на практике такой способ не очень подходит, но если вы, например, для каждого пункта меню зададите атрибут id, то сможете обратиться к нему так:

```
$(this).attr("id")
```

Полный код:

```
<script type="text/javascript">
$(function(){
  $("ul.sf-menu").superfish({
    animation: { height: "show" },
    delay: 1200,
    onShow: function(){ alert("test"); }
  });
});
</script>
```

Остальные опции, которые вы можете установить для Superfish-меню, приведены в таблице 9.1.

Таблица 9.1. Опции плагина Superfish

Параметр	Описание
animation	Позволяет указать анимационный объект. Принимает те же значения, что и первый аргумент метода animate() в jQuery
autoArrows	Если этот параметр равен true, то автоматически генерируются стрелки в разметке
delay	Задержка отображения меню после выхода указателя мыши за его пределы. Задается в миллисекундах, значение по умолчанию - 800
disableHI	Если этот параметр равен true, то плагин hoverIntent не будет обнаруживаться и его возможности не будут использоваться
dropShadows	Определяет, будут ли группы пунктов подменю отбрасывать тень (true) или нет (false)
onInit	Содержит пользовательскую функцию, которая будет вызвана при инициализации меню
onBeforeShow	Функция, которая будет вызвана перед показом очередного меню
onShow	Функция, которая будет вызвана сразу после открытия очередного меню
onHide	Функция, которая будет вызвана сразу после закрытия меню
speed	Время выполнения анимации. Значение по умолчанию 'normal'. Другие допустимые параметры: 'slow' (медленная анимация) или 'fast' (быстрая анимация)

На многих сайтах вы видели не только горизонтальное, но и вертикальное меню. Плагин Superfish позволяет создать и такое меню. Это достаточно просто. Нужно добавить файл стилей superfish-vertical.css (не удаляя обычный superfish.css):

```
<link rel="stylesheet" type="text/css" href="css/superfish.css"
/>
<link rel="stylesheet" type="text/css" href="css/superfish-
vertical.css" />
```

Этот файл, как и superfish.css, входит в комплект Superfish и является стандартным. Далее при описании списка, представляющего меню, нужно указать класс sf-vertical:

```
<ul class="sf-menu sf-vertical">
...
```

Больше никаких изменений ни в код, ни в HTML-разметку вносить не стоит.

9.3. Эффектная полоска прокрутки

Наверняка на многих сайтах, особенно посвященных играм, вы видели эффектные скрол-бары (полоски прокрутки), заменяющие скучные стандартные полоски браузеров. Реализовать подобную полоску прокрутки можно с помощью jQuery-плагина под названием jScrollPane.

На сайте jScrollPane (http://jscrollpane.kelvinluck.com/) вы найдете полную информацию о том, как скачать и где просмотреть демо плагина. Действительно, на сайте приводится очень много примеров (как JavaScript-, так и CSS-код), поэтому нет смысла приводить их в книге. Все достаточно просто - скачали плагин, вызвали стандартный код и указали контент, который должен быть прокручен.

Как выглядят полоски прокрутки jScrollPane, показано на рис. 9.3. Нужно отметить, что это только один из возможных вариантов полоски прокрутки, а на сайте разработчиков вы найдете гораздо больше разных полосок - как по внешнему виду, так и по функционалу.

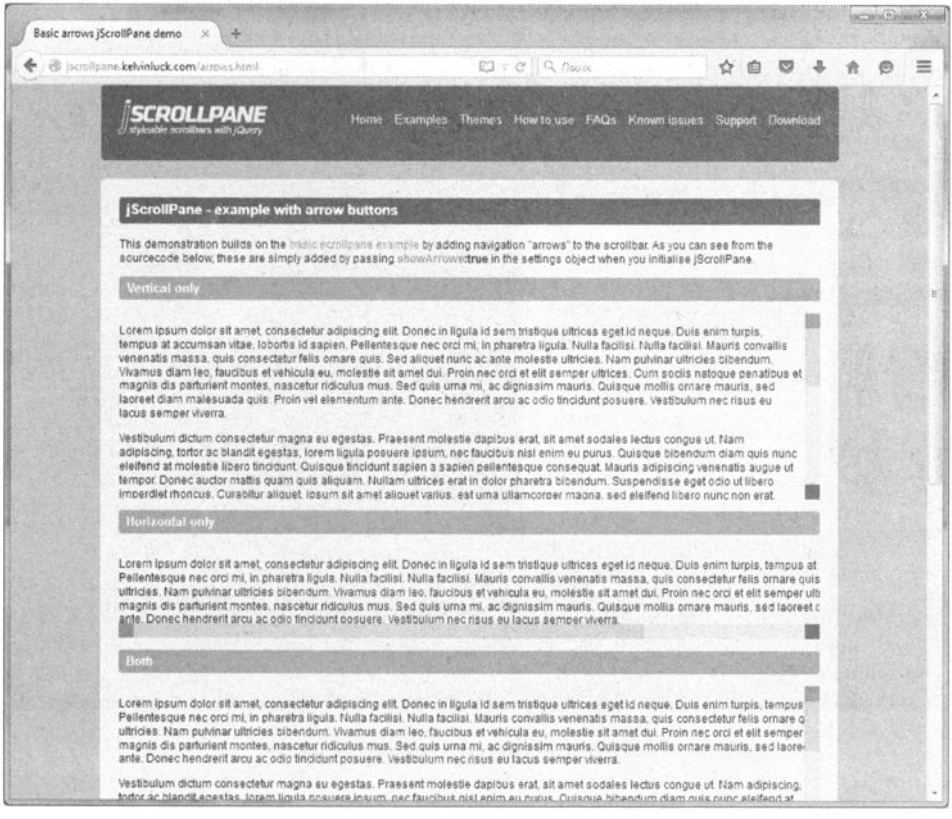


Рис. 9.3. Полоска прокрутки jScrollPane

9.4. Раздвигающееся меню

Для фан-сайтов, сайтов по продаже одежды может пригодиться раздвигающееся меню, показанное на рис. 9.4. Данное меню реализовано с помощью jQuery. Просмотреть его в действии можно по адресу:

http://tympanus.net/Tutorials/ExpandingImageMenu/

О том, как создать такое меню, вы узнаете по адресу:

http://tympanus.net/codrops/2011/03/16/expanding-image-menu

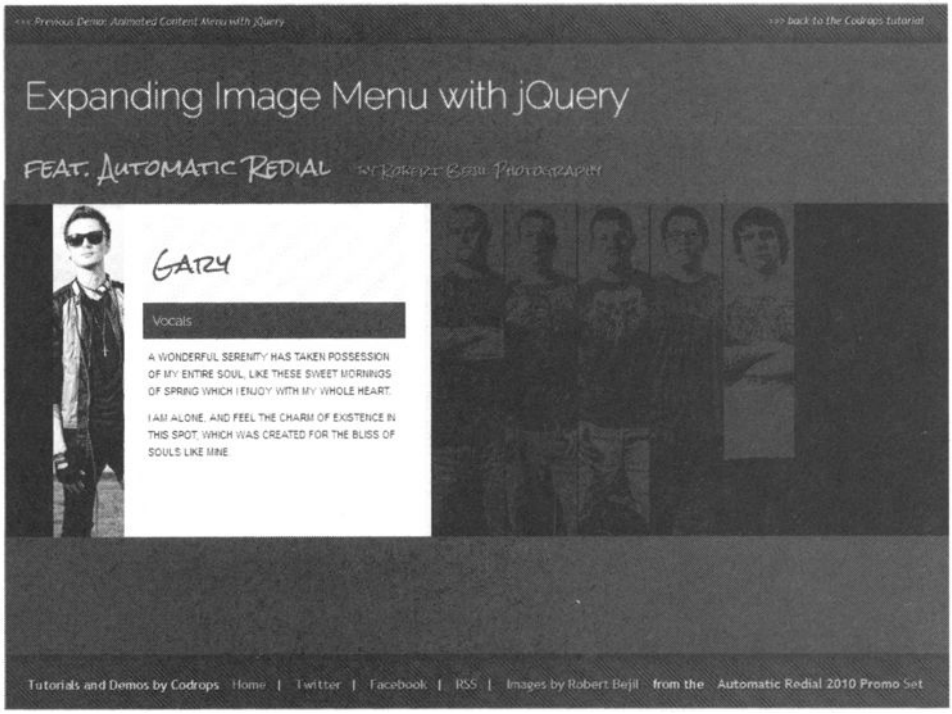
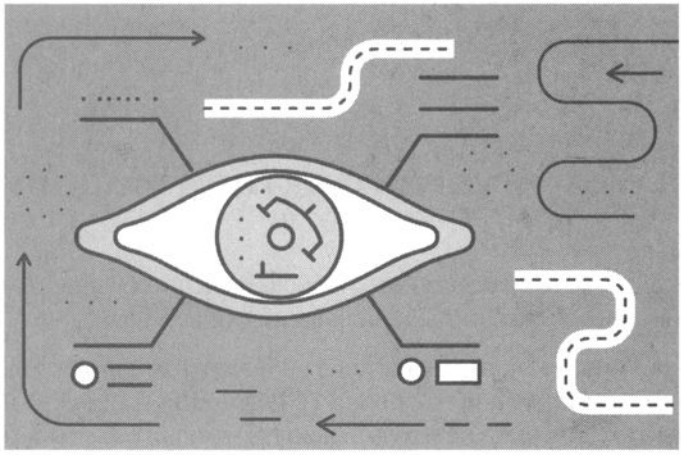
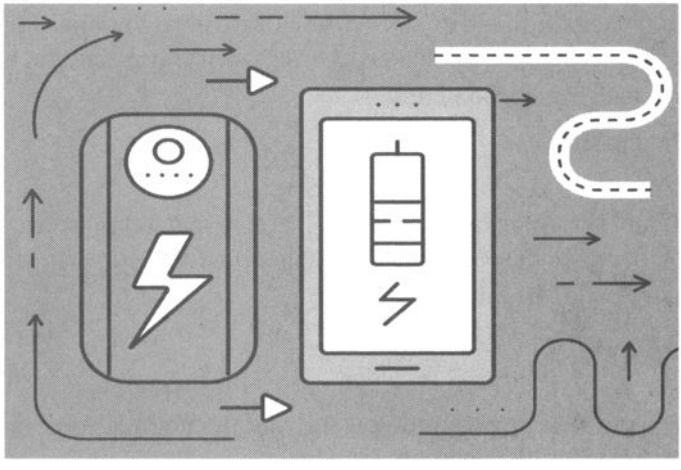


Рис. 9.4. Раздвигающееся меню



Глава 10.

Объектная модель



10.1. Структура объектной модели

Объектная модель браузера - это набор объектов, обеспечивающих доступ к содержимому веб-страницы и ряду функций веб-браузера.

Объектная модель представляется в виде иерархии объектов. Другими словами, есть объект верхнего уровня (родительский объект) и подчиненные ему объекты (дочерние объекты). У подчиненных объектов могут быть собственные подчиненные объекты и т.д. Доступ к подчиненным объектам осуществляется через точку:

```
<родительский_объект>.<дочерний_объект>.{свойство}
<родительский_объект>.<дочерний_объект>.{метод}
```

Часто объект верхнего уровня (и даже некоторые подчиненные объекты) можно не указывать. Например, мы не раз использовали метод alert() объекта window:

```
window.alert('Привет');
```

В этом случае window - объект самого верхнего уровня, представляющий веб-браузер, поэтому мы можем его не указывать, так как объект window подразумевается по умолчанию:

```
alert('Привет');
```

Зато мы не раз опускали "window" при обращении к методу document. write(). Объект document является дочерним для объекта window, поэтому правильнее было бы писать код так:

```
window.document.write('Привет');
```

Однако такая конструкция не очень удобна, поэтому мы использовали ее сокращенную форму (document.write()).

В этой главе будет рассмотрена объектная модель браузера Internet Explorer. Эта же модель полностью поддерживается браузерами Edge (преемник Internet Explorer) и Google Chrome. Именно поэтому данная и многие другие книги по JavaScript ориентируются на объектную модель IE.

10.2. Основные объекты объектной модели IE

- Кроме объекта window в объектной модели имеются следующие объекты (все они являются подчиненными объектами объекта window):
- event - с этим объектом мы уже знакомы, о нем подробно рассказывается в главе 12, где будет рассмотрена обработка событий.
- frame - используется для работы с фреймами. Сегодня использование фреймов не поощряется, поэтому этот объект мы рассматривать в книге не будем.
- history - используется для работы с историей (журналом) веб-браузера.
- navigator - предоставляет информацию о браузере.
- location - содержит URL-адрес текущей веб-страницы.
- screen - используется для доступа к характеристикам экрана компьютера пользователя.
- document - используется для доступа к документу, загруженному в браузер. Содержит следующие дочерние классы:
    - all - предоставляет доступ ко всем элементам сразу.
    - anchors - коллекция "якорей", заданных тегом <a>.
    - elements - предоставляет доступ к элементам формы.
    - forms - предоставляет доступ к формам.
    - frames - все фреймы.
    - images - предоставляет доступ ко всем изображениям.
    - links - предоставляет доступ к ссылкам.
    - scripts - предоставляет доступ к скриптам.
    - styleSheets - предоставляет доступ к стилям.

10.3. Объект window

Как уже отмечалось, объект window представляет сам браузер. Далее мы рассмотрим свойства и методы этого объекта. Основные свойства объекта window приведены в таблице 10.1.

Таблица 10.1. Основные свойства объекта window

Свойство	Описание
defaultStatus	Сообщение, выводимое по умолчанию в строке состояния.
Status	Сообщение, отображаемое в данный момент в строке состояния. В браузере IE есть строка состояния, но она может отсутствовать в других браузерах, например ее нет в Google Chrome, и любая попытка изменить установки свойства status просто ни к чему не приведет. `window.status = "Новое состояние";`
Length	Число фреймов в данном окне. Как уже отмечалось, сегодня с фреймами уже никто не работает, поэтому и мы не будем.
Parent	Ссылка на родительское окно
Self	Ссылка на текущее окно
Top	Ссылка на самое верхнее родительское окно
Opener	Ссылка на окно, которое открыло данное окно
Closed	Если равно true, то данное окно открыто, если false - закрыто (хотя по названию свойства можно подумать, что все наоборот)
screenLeft	Горизонтальная координата левого верхнего угла окна. В Firefox вместо этого свойства используется свойство screenX
screenTop	Вертикальная координата левого верхнего угла окна. В Firefox вместо этого свойства используется свойство screenY
clientInformation	Объект navigator, то есть сам браузер

С объектом window связаны следующие события:

- onload - происходит после загрузки веб-страницы.
- onunload - происходит перед выгрузкой документа.
- onscroll - происходит при прокручивании содержимого окна или фрейма.
- onresize - при изменении размеров окна.
- onblur - когда окно теряет фокус.
- onfocus - когда окно получает фокус.
- onerror - вызывается, когда в коде JavaScript возникает ошибка. В качестве обработчика этого события указывается функция, которой передается три параметра - описание ошибки, URL и номер строки. Функция должна вернуть true, если ошибка обработана, и false в противном случае.

Краткое описание методов объекта window приведено в таблице 10.2. Далее мы рассмотрим эти методы подробнее.

Таблица 10.2. Методы объекта window

Метод	Описание
alert()	Отображает окно сообщения
confirm()	Выдает окно подтверждения (см. гл. 8)
prompt()	Отображает окно с полем ввода (см. гл. 8)
showModalDialog()	Отображает модальное диалоговое окно
open()	Открывает новое окно веб-браузера
close()	Закрывает текущее окно
blur()	Снимает фокус с окна
focus()	Переносит фокус на текущее окно и генерирует событие onfocus
navigate(URL)	Загружает указанный URL в текущем окне браузера. В Firefox нет метода navigate()

stop()	Прерывает загрузку страницы. Этот метод есть в других браузерах, но его нет в браузере IE
resizeBy(x, y)	Изменяет размеры окна
resizeTo(width, height)	Устанавливает новые размеры окна
moveBy(x, y)	Перемещает окно на x пикселей вправо, на y пикселей вниз
scrollBy(x, y)	Прокручивает окно на заданное расстояние
scrollTo(x, y)	Прокручивает окно в точку с заданными координатами x, y
setTimeout()	Создает таймер. Далее будет продемонстрировано использование этого метода

В таблице 10.2 приведены далеко не все методы объекта window, а только основные. Остальные методы будут рассмотрены по мере необходимости в этой главе.

10.3.1. Метод open(): создаем новые окна

Метод open() используется для открытия нового окна браузера и загрузки в него веб-страницы. Синтаксис следующий:

```
[var <окно> = ]window.open(<URL>, [<Имя окна>], [<Свойства
>]);
```

Обязательным является только параметр `URL`, задающий адрес страницы, который нужно открыть в окне. Необязательный параметр `Имя окна` задает имя создаваемого окна, а параметр `Свойства` - свойства нового окна. При желании можно создать переменную `окно`, которую можно использовать для управления окном. Свойства окна (задаются последним параметром) приведены в таблице 10.3.

Таблица 10.3. Свойства нового окна (метод open())

Свойство	Описание
left, top	Задают горизонтальную и вертикальную координаты левого верхнего угла нового окна

width, height	Ширина и высота создаваемого окна
Fullscreen	Если это свойство равно yes или 1, тогда создаваемое окно будет открыто в полноэкранном режиме. Если свойство равно no или 0, режим окна будет обычным
Resizable	Если это свойство равно yes или 1, тогда у нового окна будет возможность изменения размера. Если свойство равно no или 0, размер окна нельзя будет изменить
Location	yes или 1 - у нового окна будет строка адреса no или 0 - у нового окна не будет адресной строки
Menubar	yes или 1 - у нового окна будет строка меню no или 0 - у нового окна не будет меню
Scrollbars	yes или 1 - у нового окна будут полоски прокрутки no или 0 - у нового окна не будет полосок прокрутки
Status	yes или 1 - у нового окна будет строка состояния no или 0 - у нового окна не будет строки состояния
Titlebar	yes или 1 - у нового окна будет заголовок no или 0 - у нового окна не будет заголовка
Toolbar	yes или 1 - у нового окна будет заголовок no или 0 - у нового окна не будет заголовка

Пример использования метода window.open() приведен в листинге 10.1. При нажатии на кнопку **Open window** происходит вызов функции wopen(), которая открывает окно с сайтом автора этой книги и возвращает дескриптор окна. При вызове window.open() мы указываем URL, название сайта и строку, описывающую опции окна. В данном случае мы не будем отображать строку состояния и панель инструментов.

Листинг 10.1. Использование метода window.open()

```
<html>
<head>
  <title>Window</title>
  <script>

   function wopen() {
```

```
    var options = "status = no,toolbar = no";
    return window.open("http://microsoft.com", "Сайт Microsoft", options)
    }

  </script>
</head>
<body>
  <input type="button" onclick="wopen();" value="Открыть
окно">
</body>
</html>
```

10.3.2. Метод showModalDialog()

Ранее были рассмотрены методы alert(), prompt() и confirm(). Они были рассмотрены раньше времени, чтобы заинтересовать вас и продемонстрировать возможности JavaScript. Но есть и еще один метод, который не был рассмотрен раньше, - метод showModalDialog(). Данный метод используется для отображения модальных окон, которые могут заменить окна, отображаемые методом prompt().

Синтаксис следующий:

```
[var <окно> = ]window.showModalDialog(<URL>,
[<Аргументы>],[<Свойства>]);
```

Первый параметр задает URL страницы, которая будет загружена в модальное окно. Второй параметр позволяет передать в окно произвольный набор аргументов. Третий параметр определяет внешний вид окна. Свойства модального окна (третий параметр) приведены в таблице 10.4.

Таблица 10.4. Свойства модального окна

Свойство	**Описание**
dialogWidth dialogHeight	Задает ширину и высоту окна
dialogTop, dialogLeft	Определяют вертикальную и горизонтальную координаты
Edge	Задает вид границы окна: вдавленный (sunken) или выпуклый (raised)

Resizable	Будет (yes, 1) ли возможность изменить размеры окна или нет (no, 0)
Scroll	Будут (yes, 1) отображаться полосы прокрутки или нет (no, 0)
Status	Будет (yes, 1) отображаться строка состояния или нет (no, 0)
dialogArtuments	Переменная или массив переменных, передаваемых в окно
Border	Задает толщину рамки (thick или thin)
Center	Выравнивает (yes) окно по центру экрана
returnValue	Возвращаемое окном значение (см. далее)

Теперь рассмотрим небольшой пример. Пусть у нас будет основная страница index.html (лист. 10.2) и страница modal.html (лист. 10.3), которая будет загружена в модальное окно, вызываемое сценарием из страницы index.html. Страница index.html будет содержать кнопку **Открыть диалог**, при нажатии которой откроется окно, в которое нужно ввести имя и фамилию.

Листинг 10.2. Страница index.html

```
<html>
<head>
<meta charset="utf-8">
<title>Модальный диалог</title>
<script>
function wmopen() { // Открываем модальное окно
  var obj = window.showModalDialog("modal.html", ["Fitstname",
"Lastname"],
            "dialogWidth:300px; dialogHeight:200px;
center=yes; status=no;");
  if (obj != null) {
    var msg = "Имя: " + obj.first;
    msg += "<br>Фамилия: " + obj.last;
    document.getElementById("div1").innerHTML = msg;
  }
}
</script>
<body>

<p><input type="button" value="Открыть диалог"
onclick="wmopen();"></p>
<div id="div1"></div>
```

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

Листинг 10.3. Страница modal.html

```
<html>
<head>
<meta charset="utf-8">
<title>Введите имя и фамилию</title>
<script>
function Onclick() {
  var o = {};
  o.first = document.forms[0].first.value;
  o.last = document.forms[0].last.value;
  window.returnValue = o;
  window.close();
}
function Onload() {
  document.forms[0].first.value = window.dialogArguments[0];
  document.forms[0].last.value = window.dialogArguments[1];
}
</script>
<body onload="Onload();">

<form action="" id="frm">
<div style="text-align: center">
Имя:<br><input type="text" name="first"><br>
Фамилия:<br><input type="text" name="last"><br>
<input type="reset" value="Очистить">
<input type="button" value="OK" onclick="Onclick();">
</div>

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

А теперь разберемся, что есть что. Сначала в index.html при нажатии кнопки Открыть диалог мы открываем модальное окно:

```
var obj = window.showModalDialog("modal.html", ["Firstname",
"Lastname"],
            "dialogWidth:300px; dialogHeight:200px;
center=yes; status=no;");
```

Первый параметр метода showModalDialog - это документ, который будет загружен в открытое окно. Объекту окна будут переданы два параметра, значение первого параметра - "Firstname", значение второго - "Lastname". Третий параметр - это параметры диалога. Мы задаем ширину и высоту диалога (300 и 200 пикселей соответственно), указываем центрирование и отключаем строку состояния. Подробнее о внешнем виде мы поговорим чуть позже.

Теперь переходим к файлу modal.html (к index.html мы еще вернемся). Для обработки события onload используется функция Onload(). Она получает параметры, переданные диалогу, и заполняет поля формы - имя и фамилию. В принципе, без этого можно было бы обойтись, но зато теперь вы знаете, как получить параметры окна и как их можно использовать.

При нажатии кнопки OK вызывается функция Onclick. Данная функция формирует объект o, который будет отправлен в качестве возвращаемого значения вызывающему окну (то есть сценарию в index.html). После того как значение свойства returnValue сформировано, вызывается метод close(), чем модальное окно закрывается.

Теперь о внешнем виде. Посмотрите на рис. 10.1. На нем изображено созданное нами окно в браузерах Internet Explorer, Firefox (слева направо). Наиболее корректно окно отображается в Internet Explorer: у него нет строки адреса (а зачем она нужна модальному окну?), окно всплывает точно по

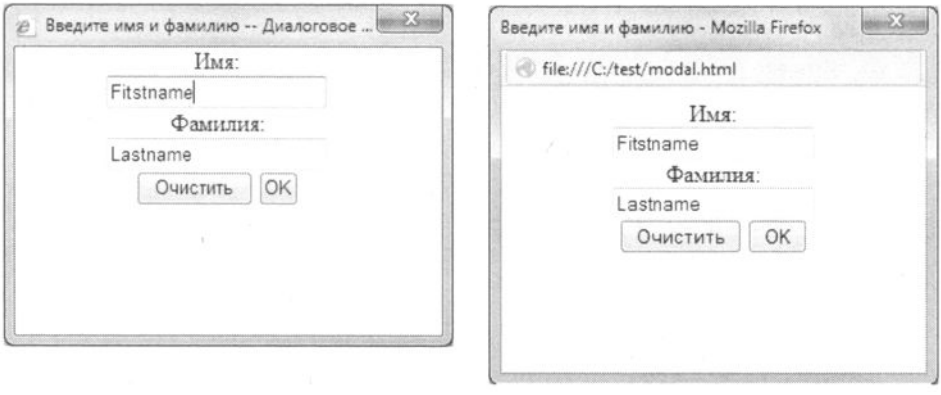


Рис. 10.1. Внешний вид модального окна в разных браузерах

центру экрана. У Chrome появилась строка адреса, убрать которую у меня не получилось, придется с ней мириться. Зато окно всплывает точно по центру экрана. А в Firefox мало того, что окно (несмотря на center=yes) всплывает в верхнем левом углу, так еще и тоже отображается строка адреса, которую всем пользователям видеть как бы и не обязательно.

Возвращаемся к index.html. Итак, объект передан. Если переданный объект не равен null, тогда мы формируем сообщение, которое будет выведено в качестве значения div1 нашей HTML-страницы.

```
if (obj != null) {
  var msg = "Firstname: " + obj.first;
  msg += "<br>Lastname: " + obj.last;
  document.getElementById("div1").innerHTML = msg;
}
```

Модальные диалоговые окна, в отличие от метода prompt(), очень удобно использовать, когда пользователю необходимо ввести несколько значений, например имя и фамилию, или имя пользователя и пароль.

Однако с модальными окнами и методом showModal наблюдается некоторая проблема - этот метод уже не поддерживается в последних версиях браузеров Chrome и Opera. Можно предположить, что со временем и другие браузеры (IE, Firefox) откажутся от него. Что же делать?

Выход как всегда есть, и это HTML5-диалоги. К преимуществам таких диалогов можно отнести:

- Они описываются непосредственно в HTML-коде, вам не нужен дополнительный файл (в нашем случае modal.html) для хранения кода диалога.
- Не нужны сложные конструкции для доступа к полям диалога. Поскольку поля описываются в пределах того же документа, что и вызывает диалог, доступ к его полям осуществляется так же, как в случае с обычной формой.
- Используя CSS, вы можете настроить внешний вид диалога так, как вам нравится.

Недостаток (точнее особенность, недостатком это сложно назвать) у HTML5-диалога только один - он появляется в том месте страницы, где был объявлен. На рис. 10.2 диалог был объявлен в начале страницы, поэтому он и появился вверху окна браузера. Чтобы задать другое положение, нужно настраивать стили (CSS).

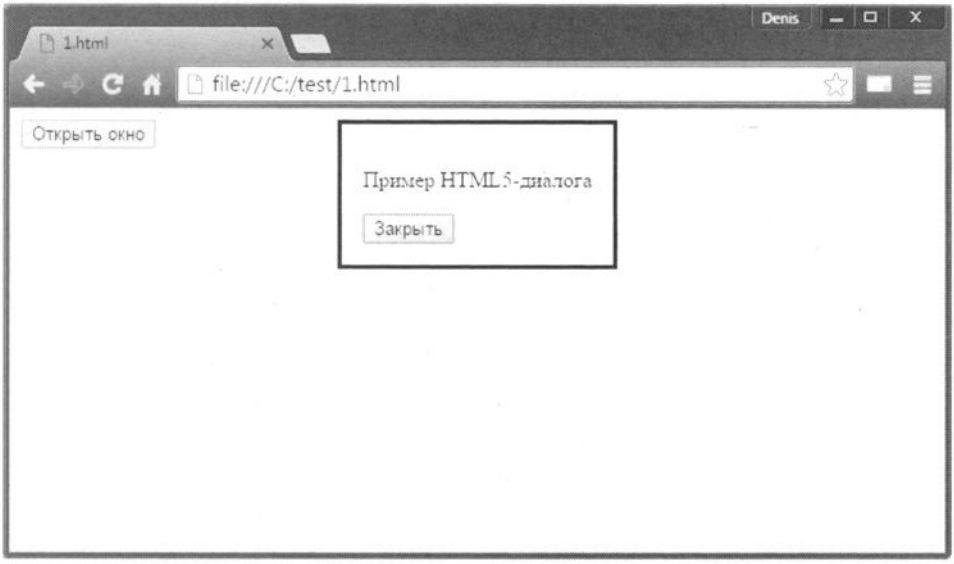


Рис. 10.2. HTML5-диалог

В листинге 10.4 приводится код страницы, отображающей этот диалог.

Листинг 10.4. HTML5-диалог

```
<html>
<head>
  <meta charset="utf-8">
  <script>
  function Onclick(){
      var dialog = document.getElementById('Dialog');
    document.getElementById('showDialog').onclick = function()
{
      dialog.show();
    };
    document.getElementById('closeDialog').onclick =
function() {
      dialog.close();
    };
  };
  </script>
  </head>
  <body>
    <dialog id="Dialog">
      <p>Пример HTML5-диалога</p>
      <button id="closeDialog">Закрыть</button>
```

```
    </dialog>
    <button id="showDialog" onclick="Onclick();">Открыть
окно</button>
  </body>
</html>
```

10.3.3. Метод setTimeout()

Метод setTimeout() используется для управления таймерами, а именно setTimeout() можно использовать для однократного выполнения определенной функции в заданное время. Вы задаете интервал времени, по истечении которого будет выполнена указанная функция.

Кроме этого метода у объекта window есть также и другие методы, касающиеся таймеров:

- clearTimeout(<таймер>) - сбрасывает таймер, установленный методом setTimeout().
- setInterval() - создает таймер, который многократно выполняет указанную функцию или выражение через заданный интервал времени.
- clearInterval(<интервал>) - сбрасывает интервал, установленный методом setInterval().

Рассмотрим несколько примеров. Установить таймер можно так:

```
var timer = setTimeout(<функция или выражение>, <инвервал>);
```

Сбросить таймер timer можно так:

```
clearTimeout(timer);
```

Метод setInterval() вызывается так:

```
var intr = setInterval(<функция или выражение>, <инвервал>);
```

Как использовать таймеры? Первое, что приходит в голову, - организовать часы на страничке. В листинге 10.4 функция Timerstart() запускает таймер посредством setInterval(). Каждую секунду (1000 мс) будет запускаться функция DisplayTime(), отображающая время. Для остановки таймера используется функция Timerstop().

Листинг 10.5. Пример использования таймеров

```
<html>
<head>
<title>Timers</title>
<script>

var clock;

function Timerstart() { // Запускает таймер
clock = setInterval("DisplayTime();", 1000);
}

function DisplayTime() { // Отображает время
var d = new Date();
var CurrTime = (d.getHours()<10) ? "0" : "";
CurrTime  += d.getHours();
CurrTime  += (d.getMinutes()<10) ? ":0" : ":";
CurrTime  += d.getMinutes();
CurrTime  += (d.getSeconds()<10) ? ":0" : ":";
CurrTime  += d.getSeconds();
document.getElementById("div1").innerHTML = CurrTime;
}
function Timerstop() { // Останавливает таймер
clearInterval(clock);
}

</script>
</head>
<body onload="Timerstart();">
<div id="div1"></div>

<input type="button" value="Start" onclick="Timerstart();">
<input type="button" value="Stop" onclick="Timerstop();">

</body>
</html>
```

10.4. Объект navigator: получение информации о браузере и системе

Объект navigator можно использовать для получения информации о веб-браузере. Свойства объекта navigator содержат много полезной информации:

- appName – имя Web-браузера;
- appCodeName – кодовое имя версии Web-браузера;
- appVersion – версия Web-браузера;
- appMinorVersion – вторая цифра в номере версии Web-браузера;
- userAgent – комбинация свойств appCodeName и appVersion;
- cpuClass – тип процессора компьютера пользователя;
- platform – название клиентской платформы;
- systemLanguage – код языка операционной системы пользователя;
- browserLanguage – код языка Web-браузера;
- userLanguage – код языка Web-браузера;
- onLine –true, если клиент в настоящее время подключен к Интернету, и false, если мы - оффлайн;
- cookieEnabled – режим работы cookie: возвращает true, если прием cookie разрешен.

Все эти свойства вы можете использовать для получения различной информации о браузере. Например,

```
document.write(navigator.userAgent);
```

В браузере Google Chrome выведет следующую строку:

Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36

Все эти свойства можно использовать, например, для отображения информации на языке клиента. Допустим, вы пишете сценарий, который должен "общаться" с пользователем на русском или английском. С помощью свойств объекта navigator можно попытаться определить язык браузера/системы и отобразить информацию на этом языке. Если язык браузера/системы не поддерживается вашим сценарием, тогда будете отображать информацию на английском. Это как один из вариантов использования свойств объекта navigator. Единственный недостаток - не все свойства поддерживаются отдельными браузерами. Однако свойства вроде AppName и userAgent поддерживаются всегда, поэтому их можно использовать для определения браузера и написания более кроссбраузерного кода, адаптированного под тот или иной браузер, например:

```
var browser = navigator.userAgent;
if (browser.indexOf("Explorer") != -1) {
   // код для IE
}
if (browser.indexOf("Firefox") != -1) {
   // код для Firefox
}
if (browser.indexOf("Opera") != -1) {
   // код для Opera
}
if (browser.indexOf("Chrome") != -1) {
   // код для IE
}
```

10.5. Объект screen: информация о мониторе пользователя

Объект screen хранит информацию о мониторе пользователя. Получить доступ к этой информации можно посредством свойств этого объекта:

- width —ширина экрана в пикселах;
- height —высота экрана в пикселах;
- availWidth — ширина, доступная для окна браузера;
- availHeight — высота, доступная для окна браузера;
- colorDepth — глубина цвета: 4 — для 16 цветов, 8 — для 256 цветов, 32 — для 16,7 млн цветов.

10.6. Объект location: строка адреса браузера

Объект location предоставляет информацию об URL-адресе текущей страницы. Получить доступ к этой информации, как обычно, можно через свойства этого объекта:

- href — полный URL-адрес документа;
- protocol — идентификатор протокола;
- port — номер порта;
- host — имя сервера вместе с номером порта;
- hostname — имя компьютера в сети Интернет;
- pathname — путь и имя файла;
- search — строка параметров, указанная после знака "?" (включая этот знак);
- hash — строка, указанная после знака "#" (включая этот знак).

В отличие от предыдущих двух объектов у объекта location есть также и методы. Метод assign() загружает указанный URL, reload() - перезагружает документ, replace() - загружает указанный в качестве параметра URL, а информация о предыдущем адресе удаляется из объекта history.

Однако загрузить новый документ можно не только с помощью метода assign(), но и путем изменения свойства href, например:

```
windows.location.href = "http://dkws.org.ua";
```

10.7. Объект history: список истории

Объект history позволяет получить доступ к списку всех ранее просмотренных веб-страниц. Свойство length содержит размер списка истории. Методы этого объекта следующие:

- go(<номер>) - переходит в списке истории на позицию с указанным номером.
- back() - загружает в окно браузера предыдущий документ, можно использовать для реализации кнопки Back.

- forward() - загружает в окно браузера следующий документ из списка истории. Можно использовать для реализации кнопки Next.

С помощью методов back() и forward() можно создать кнопки или ссылки Back/Next. Вот пример создания таких ссылок:

```
<p><a href="javascript:history.back();">Back</a></p>
<p><a href="javascript:history.forward();">Next</a></p>
```

10.8. Объект document: обращение к элементам документа

Объект document используется для обращения к элементам документа. Свойств у этого объекта довольно много, поэтому для лучшего восприятия они приведены в таблице 10.5.

Таблица 10.5. Свойства объекта document

Свойство	Описание
activeElement	Ссылка на активный элемент документа
documentElement	Ссылка на тег <html>
Body	Ссылка на все содержимое тега <body>
Title	Название документа, указанное в теге <title>
URL	Адрес документа
Referrer	Адрес страницы, с которой посетитель перешел на эту страницу
parentWindow	Окно, которому принадлежит документ
Cookie	Используется для хранения данных на компьютере пользователя, например можно запомнить выбранную пользователем тему оформления
readyState	Состояние документа. Оно может быть следующим: uninitialized - не инициализирован, недоступен. loading - документ загружается. interactive - загружен не полностью, но уже доступен для просмотра. complete - полностью загружен.

Location	Объект location, рассмотренный ранее
Selection	Объект selection, который будет рассмотрен чуть позже
fileCreatedDate	Дата создания файла документа (в виде строки)
fileModifiedDate	Дата последнего изменения
fileUpdatedDate	Дата обновления файла в кэше компьютера пользователя
lastModified	Дата и время последнего изменения документа
Filesize	Размер файла
bgColor	Цвет фона документа
fgColor	Цвет текста страницы
linkColor	Цвет гиперссылок документа
alinkColor	Цвет активных ссылок
vlinkColor	Цвет посещенных ссылок

Объект document поддерживает следующие методы:

- write(<текст>) - выводит текст, заданный параметром, в текущее место документа.
- writeln(<текст>) - выводит текст, заданный параметром, после чего выводит символы возврата каретки и новой строки (\r\n). Данные символы полностью игнорируются браузером, поэтому результат будет такой же, что и у метода write().
- getElementById(<ID элемента>) - возвращает ссылку на элемент с указанным ID.
- getElementByName(<название элемента>) - возвращает ссылку на элемент по его имени.
- elementFromPoint(<x>, <y>) - возвращает ссылку на элемент, который находится по координатам <x>, <y>

У каждого элемента на странице есть свои свойства и методы. В таблице 10.6 представлены свойства, общие для всех элементов.

Таблица 10.6. Основные свойства элемента веб-страницы

Свойство	Описание
All	Ссылка на коллекцию дочерних элементов
Id	Имя элемента, заданное параметром id
className	Имя класса, заданное параметром class
sourceIndex	Порядковый номер элемента, который можно использовать для ссылки на элемент из коллекции all
tagName	Имя тега элемента
parentElement	Ссылка на родительский элемент
Length	Число элементов в коллекции
height и width	Высота и ширина элемента
clientHeight и clientWidth	Высота и ширина элемента без учета рамок, границ и полосок прокрутки
clientLeft	Смещение левого края элемента относительно левого края родительского элемента без учета рамок, границ и т.д.
clientTop	Смещение верхнего края элемента относительно левого края родительского элемента без учета рамок, границ и т.д.
offsetHeight и offsetWidth	Высота и ширина элемента относительно родительского элемента
offsetLeft	Смещение левого края элемента относительно левого края родительского элемента
offsetParent	Ссылка на родительский элемент, относительно которого определяются свойства offsetHeight, offsetWidth, offsetLeft и offsetTop
innerText	Содержимое элемента без HTML-тегов. Если этому свойству присвоить новое значение, то изменится значение элемента

outerText	Содержимое элемента без HTML-тегов. Если присвоить свойству новое значение, то содержимое элемента заменится новым и будет изменен сам элемент
innerHTML	Содержимое элемента вместе с HTML-тегами. Если этому свойству присвоить новое значение, то изменится значение элемента
outerHTML	Содержимое элемента вместе с HTML-тегами
scrollHeight и scrollWidth	Высота и ширина содержимого элемента
scrollLeft и scrollTop	Положение горизонтальной и вертикальной полос прокрутки

В таблице 10.7 приведены методы, общие для всех элементов страницы.

Таблица 10.7. Основные методы элемента веб-страницы

Метод	Описание
getAdjancentText (<местонахождение>)	Возвращает текстовую строку в зависимости от указанного местонахождения
insertAdjancentHTML (<местонахождение>, <текст>)	Вставляет текст в место, заданное местонахождением
getAttribute (<имя параметра>, true | false)	Возвращает значение параметра, который задан <именем параметра>. Если второй параметр равен false, то поиск параметра тега происходит без учета регистра символов
setAttribute (<имя параметра>, <значение>, true | false)	Присваивает <Значение> параметру, который задан <именем параметра>. Если третий параметр равен false, то поиск параметра тега происходит без учета регистра символов
removeAttribute(<Имя параметра>, true | false)	Удаляет параметр тега текущего элемента. Если второй параметр равен false, то поиск параметра тега происходит без учета регистра символов

clearAttributes()	Удаляет все параметры тега элемента, кроме параметров id и name
contains(<имя>)	Возвращает true, если элемент с этим именем содержится внутри текущего элемента

Используя все эти свойства и методы, вы можете манипулировать элементами страницы. Например, вы можете обратиться к любому изображению и получить его источник (заданный параметром тега src):

```
document.images[индекс].src
```

Нумерация, как обычно, начинается с 0. В цикле можно обойти все изображения (как и другие элементы страницы) с целью получения информации о них.

10.9. Объект style: доступ к таблице стилей

Объект style используется для получения доступа к каскадным таблицам стилей (CSS). Свойства этого объекта соответствуют атрибутам в CSS с небольшими отличиями: удаляются символы "-", а первые буквы всех слов в названии атрибута, кроме первого, делаются прописными. Пример преобразования имен:

```
color = color
font-size = fontSize
```

В листинге 10.5 приведен пример изменения цвета текста в с ID div1.

Листинг 10.5. Изменение стиля с помощью JavaScript

```
<html>
<body onload="OnLoad();">
<script>

function OnLoad() {
var div1 = document.getElementById("div1");
div1.style.color = 'red';
```

```
}

</script>
<div id="div1" >Привет</div>
</body>
</html>
```

10.10. Объект selection: работа с выделением

Объект selection используется для работы с выделенным фрагментом на странице. Свойство type объекта selection возвращает значение None, если ничего не выбрано, и Text, если выбран текст.

Методы у объекта selection следующие:

- clear() - стирает выделенный текст.
- empty() - снимает выделение текста.

Метод getSelection() объекта window возвращает объект Selection. У него есть различные свойства и методы, но самый полезный из них - метод toString(), который возвращает выделение в виде строки. Данный метод будет работать только в браузерах Chrome, Firefox и Opera. В IE код получения выделения нужно реализовывать через диапазоны. Рассмотрим сценарий 10.6. В нем приведен простой код получения выделения, который будет работать в браузерах Chrome, Firefox и Opera.

Листинг 10.6. Получение выделения текста

```
<html>
<head>
  <title>Selection</title>
</head>
<body>
<script>

function OnClick() {
 var selText = window.getSelection().toString();
 window.alert(selText);

}
```

```
</script>
  <div id="div1" >Выделите текст и нажмите кнопку</div>
  <p><input type="button" onclick="OnClick();" value="Нажми
меня">
</body>
</html>
```

Логика проста: мы получаем выделение методом getSelection(), преобразуем его в текст методом toString() и отображаем методом alert. Когда же нужно обеспечить совместимость с браузером IE (а это нужно сделать, потому что все еще этот браузер занимает значительную долю на рынке браузеров), то код будет несколько сложнее. Нужно переписать функцию OneClick(), которая приведена в листинге 10.7. Остальной код будет таким же.

Листинг 10.7. Универсальный код, работает в IE, Chrome, Firefox и Opera

```
function OnClick() {

if (window.getSelection) {
  window.alert(window.getSelection().toString());
}
else { // For IE
  if (document.selection.type=="Text") {
  var range1 = document.selection.createRange();
  window.alert(range1.text);
  document.selection.empty();
}
else {
  window.alert("No selected text");
  }
} // IE block

} //function
```

10.11. Полезные примеры

В качестве полезных примеров мы рассмотрим добавление вашего сайта в Избранное и в качестве домашней страницы, а также работу с Cookies.

10.11.1. Добавление сайта в Избранное

Наверное, вы видели на многих сайтах ссылки, позволяющие добавить сайт в избранное или установить его в качестве домашней страницы. В принци-

пе, данные действия можно совершить и с помощью средств самого браузера, однако пользователь может забыть это сделать, а создав соответствующие ссылки (кнопки) на самом сайте, вы можете ему об этом напомнить. Учтите, что добавление своего сайта в избранное или установка в качестве домашней страницы без разрешения (ведома) пользователя считается дурным тоном, поэтому даже и не думайте вызывать приведенные ниже методы из обработчика события onload вашей страницы!

Для добавления сайта в избранное используется метод addFavorite объекта external. В качестве параметров этого методу нужно передать адрес страницы и ее описание. Ниже приведена функция, реализующая добавление сайта в избранное. Вы ее можете использовать в качестве обработчика нажатия ссылки.

```
function addToFavorites() {
  external.addFavorite("http://example.com", "Example.com");
  window.alert('Спасибо!');
  return false;
}
...
<p><a href="http://example.com" onclick="return
addToFavorites();">
Добавить в Избранное</a><br>
```

10.11.2. Установка сайта в качестве домашней страницы

Чтобы установить ваш сайт в качестве домашней страницы, вы можете использовать функцию, код которой приведен ниже:

```
function setAsHomePage(obj) {
  obj.style.behavior="url(#default#homepage)";
  obj.setHomePage("http://example.com");
  window.alert('Спасибо!');
  return false;
}

...
<p><a href="http://example.com" onclick="return
setAsHomePage(this);">
Установить как домашнюю страницу</a><br>
```

10.11.3. Работа с Cookies

Веб-браузеры позволяют на пользовательском клиенте хранить небольшой объем информации, называемый Cookies. С помощью Cookies удобно хра-

нить настройки пользователя, например выбранную тему оформления, просмотренные ним товары и т.д. У Cookies, конечно, есть свои достоинства и недостатки, как у всего в этом мире. К достоинствам можно отнести то, что вам не нужно хранить эту не очень важную информацию на своем сервере. Это снижает нагрузку на сервер и экономит дисковое пространство на нем.

А вот недостатков больше. Доступ к Cookies может получить любой другой сценарий, запущенный на компьютере пользователя (даже если он не устанавливал эти Cookies), поэтому там нельзя хранить конфиденциальную информацию, например признак аутентификации, номера кредитных карточек, пароли и т.д. К тому же, если пользователь запустит другой браузер на этом компьютере или перейдет на другой компьютер, данные, сохраненные в Cookies, станут недоступными.

Прежде чем приступить к использованию Cookies, нужно проверить, поддерживает ли их браузер клиента:

```
if (navigator.cookieEnabled) {
// Cookies поддерживаются, можно устанавливать и читать их
}
```

Установить Cookies можно путем присвоения значения свойству cookie объекта document в следующем формате:

```
document.cookie = "<Имя>=<Значение>; [expires=<Дата>;]
[domain=<Имя домена>;] [path=<Путь>;] [secure;]";
```

В самом простом случае нужно указать пару <Имя>=<Значение>. В чуть более сложном - дату истечения срока действия. Дату нужно указывать только в таком формате:

```
Mon, 03 Feb 2014 00:00:01 GMT
```

По истечении данной даты Cookie будет удален. Получить дату в этом формате (чтоб не создавать ее вручную) можно с помощью метода setTime() и метода toGMTString(). Пример:

```
var d = new Date();
d.setTime(d.getTime()+36000000);  // Cookie будет жить 10
часов
var End_Date = d.toGMTString();   // Дата удаления cookies
```

Считать Cookie можно посредством обращения к document.cookie. Возвращенная строка будет содержать все установленные Cookie в формате "имя1=значение1; имя2=значение2".

Чтобы удалить Cookie, нужно установить его с истекшей датой. Другого способа, увы, нет.

Для облегчения работы с Cookies мною были разработаны функции, представленные в листинге 10.8. Используя их, вы можете легко установить, прочитать и удалить Cookie.

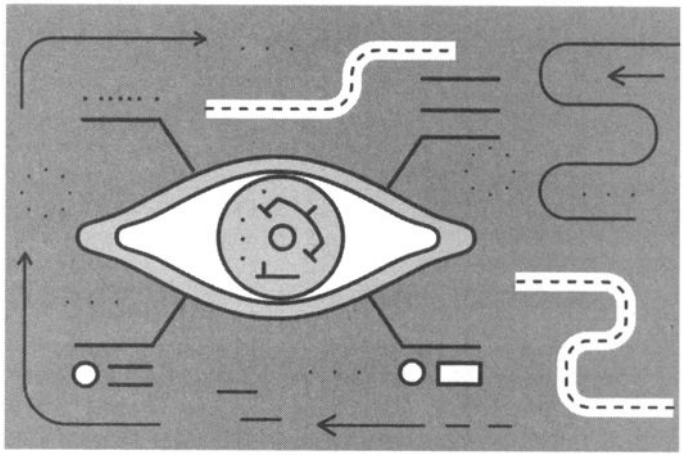
Листинг 10.8. Функции для работы с Cookies

```
function setCookie(name, value, expires, path, domain, secure)
{
    if (!name || !value) return false;
    var str = name + '=' + encodeURIComponent(value);

    if (expires) str += '; expires=' + expires.toGMTString();
    if (path)    str += '; path=' + path;
    if (domain)  str += '; domain=' + domain;
    if (secure)  str += '; secure';

    document.cookie = str;
    return true;
}

function getCookie(name) {
    var pattern = "(?:; )?" + name + "=([^;]*);?";
    var regexp  = new RegExp(pattern);

    if (regexp.test(document.cookie))
    return decodeURIComponent(RegExp["$1"]);

    return false;
}

function deleteCookie(name, path, domain) {
    setCookie(name, null, new Date(0), path, domain);
    return true;
}
```

Использовать эти функции можно так:

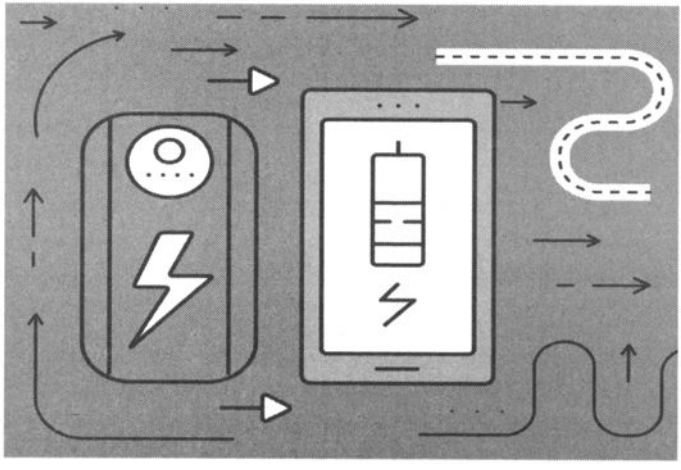
```
// устанавливаем Cookies без параметров, только имя и значения
setCookie('color', 'red');
// читаем Cookie
var color = getCookie('color');
// Выводим Cookie
window.alert(color);
```

На этом данная глава заканчивается, а в следующей мы поговорим о том, как работать с формами в JavaScript.



Глава 11.

Работа с формами в JavaScript



11.1. Коллекция Forms

Представим, что в нашем документе есть форма form1 с текстовым полем firstname. Обратиться к этому полю можно через коллекцию forms:

```
document.forms["form1"].firstname.value
```

При желании к форме можно обратиться и напрямую, что делает код компактнее:

```
document.form1.firstname.value
```

Существует и третий способ обращения к форме - по индексу:

```
document.forms[0].firstname.value
```

Нумерация форм начинается с 0. Если в документе только одна форма, то ее номер - 0.

Получить доступ к элементу независимо от того, находится ли он внутри формы или нет, можно с помощью метода getElementById() объекта document:

```
document.getElementById("firstname").value
```

Элементы формы также доступны через коллекцию elements, например:

```
document.forms["form1"].elements["firstname"].value
document.forms["form1"].elements[0].value
document.forms[0].elements[0].value
```

Используйте тот метод обращения к элементам формы, который вам больше нравится.

11.2. Свойства, методы и события объекта формы

У объекта формы есть следующие свойства:

- length — количество элементов в форме;
- action — URL-адрес сценария, который будет обрабатывать форму;
- elements — ссылка на коллекцию elements;
- encoding — MIME-тип передаваемых данных;
- method — режим пересылки данных формы программе, которая задана в action;
- enctype — метод кодирования данных формы;
- name — имя формы;
- target — имя фрейма, в который будет загружен документ.

Объект формы поддерживает всего два метода: submit() и reset(). Первый метод инициирует отправку формы на веб-сервер, второй - сбрасывает форму. Соответственно, есть два события - onsubmit и onreset. Первое происходит при отправке данных формы, второе - при сбросе формы.

11.3. Получение данных из поля ввода. Проверка правильности ввода

Сейчас мы поговорим о двух элементах формы - текстовом поле и поле для ввода пароля, которые создаются с помощью следующего HTML-кода:

```
<input type="text">
<input type="password">
```

У этих элементов формы одинаковые свойства, методы и события. Сначала рассмотрим свойства (см. табл. 11.1).

Таблица 11.1. Свойства полей ввода текста и пароля

Свойство	Описание
Value	Значение элемента формы
defaultValue	Начальное значение, задаваемое параметром value
Disabled	Если true, то поле является неактивным.
Form	Ссылка на форму
maxLength	Максимальное количество символов, которое можно ввести в поле
Name	Имя элемента
readOnly	Если это свойство равно true, то поле нельзя редактировать, если false, то можно
Type	Тип элемента формы

Методы и связанные с ними события представлены в таблице 11.2.

Таблица 11.2. Методы и свойства полей ввода текста и пароля

Метод	Событие	Описание
blur()	onblur	Убирает фокус с текущего элемента формы. При потере фокуса происходит событие onblur
focus()	onfocus, onchange	Помещает фокус на текущий элемент формы. При получении фокуса определенным элементом формы происходит событие onfocus. Событие onchange происходит при получении и изменении фокуса, изменении данных в поле или при отправке данных формы.
select()	-	Выделяет текст в поле

11.4. Работа с textarea

В отличие от обычного поля ввода текста поле textarea позволяет вводить многострочный текст. Свойства, методы и события textarea такие же, как у поля ввода. Однако у textarea нет свойства maxLenght, зато есть свойство wrap, задающее режим переноса слов. Это свойство может принимать следующее значение:

- off - слова переноситься не будут.
- physical - слова переносятся как на экране, так и при передаче данных.
- virtual - слова переносятся только на экране, но не при передаче данных на сервер.

Сейчас мы напишем простой сценарий, демонстрирующий работу с формой. Наша форма изображена на рис. 11.1. Она содержит всего три элемента - поле ввода, textarea и кнопка. При нажатии кнопки Add введенное в текстовое поле слово будет добавлено в textarea. Листинг 11.1 содержит комментарии, прочитайте их для лучшего понимания программы.

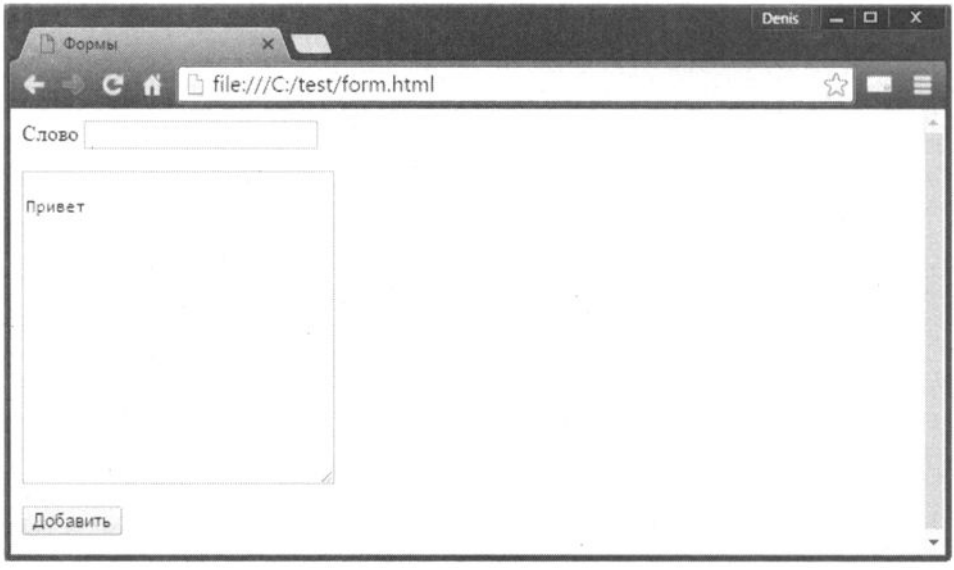


Рис. 11.1. Наша форма

Листинг 11.1. Работа с текстовыми полями

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Формы</title>
  </head>
<body>
  <script>
  function AddWord() {

  // получаем значение текстового поля
  var text1 = document.form1.text1.value;

// если текст не введен, выходим
  if (text1 == "") { windows.alert('Введите текст'); return
""; }

// получаем значение textarea
  var ta1 = document.form1.ta1.value;
  var result = ta1 + "\n" + text1;

  // новое значение textarea
  document.form1.ta1.value = result;

  // очищаем текстовое поле
  document.form1.text1.value = "";
  return text1;     // возвращает введенный текст

  }
  </script>
  <form name="form1">
    Word <input type="text" name="text1" id="text1"><br>
    <textarea name="ta1" id="ta1" cols="25" rows="15"></
textarea><br>
    <input type="button" value="Добавить"
onclick="AddWord();"><br>
  </form>
</body>
</html>
```

11.5. Работа с флажками

У флажков и переключателей несколько другой набор свойств, методов и событий. Как обычно, сначала рассмотрим свойства (таблица 11.3).

Таблица 11.3. Свойства флажков и переключателей

Свойство	Описание
checked	Если свойство равно true, переключатель или флажок находится во включенном состоянии
value	Значение текущего элемента формы
defaultChecked	Включен ли флажок или переключатель по умолчанию (true - да, false - нет)
disabled	Если true, то элемент формы выключен (его нельзя изменять)
indeterminate	Флажок находится в неопределенном состоянии (закрашен серым цветом). Возвращает false или true
form	Ссылка на форму, в которой находится элемент
name	Имя элемента
type	Тип элемента формы

Из методов флажки и переключатели поддерживают только blue() и focus(), которые были рассмотрены раньше. События тоже аналогичны рассмотренным ранее: onblur(), onfocus(). Также есть событие onclick(), которое возникает при выборе элемента.

Теперь рассмотрим небольшой пример. Пусть у нас есть группа переключателей rg1:

```
<input type="radio" name="rg1" id="radio1" value="1"
checked>Option 1
<input type="radio" name="rg1" id="radio2" value="2">Option
2<br>
<input type="radio" name="rg1" id="radio3" value="3">Option
3<br>
```

Обойти группу переключателей в цикле можно так:

```
var count = document.form1.rg1.length;
for (i=0; i<count; i++) {
      if (document.form.rg1.item(i).checked) {
            window.alert(document.form.rg1.item(i).value);
            break;
      }
}
```

После того как находим включенный переключатель, мы выводим его значение (задается атрибутом value) и прерываем цикл - остальные переключатели нет смысла просматривать, так как выбранным в группе может быть только один переключатель (не путать с флажком - checkbox!).

11.6. Работа с кнопками

Свойства кнопки подобны свойствам других элементов формы:

- value - значение кнопки (текст, который отображается на ней).
- disabled - если это свойство равно true, кнопка будет неактивной.
- form - ссылка на форму, в которой находится кнопка.
- name - имя элемента формы.
- type - тип элемента формы.

Методы также вам уже знакомы: blur() и focus(). События аналогичны переключателям: onblur(), onclick() и onfocus(). Далее приведен пример работы с кнопками. Наша форма будет содержать две кнопки - Press me и Test. При нажатии на Press me происходит инвертирование кнопки Test. То есть при одном нажатии кнопка Test будет выключена, при другом - включена и т.д.

Листинг 11.2. Пример работы с кнопками

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Формы</title>


  <script>
```

```
  function OnClick() {
    document.form1.button2.disabled = !document.form1.
button2.disabled;
  }

  </script>

  </head>
  <body>

  <form name="form1">
   <input type="button" name="button1" id="button1"
value="Нажми меня" onclick="OnClick();"><br>
   <input type="button" name="button2" id="button2"
value="Тест"><br>
  </form>
</body>
</html>
```

11.7. Проверка правильности e-mail

Очень часто перед отправкой формы на сервер требуется проверить введенные данные, например ввел ли пользователь имя, e-mail, установил ли пароль, совпадают ли пароли (если речь идет о форме регистрации). Учитывая все полученные знания, вы сами можете написать код такой проверки. Сложность может вызвать разве что проверка корректности электронного адреса. Если проверять корректность e-mail с помощью обычных проверок, то получите некомпактный и неэффективный код, перегруженный множеством операторов if, и не факт, что он будет работать правильно. Например, все мы знаем, что в любом электронном адресе должны быть символы "@" и ".", но если проверить наличие в строке только этих символов, то проверка явно не будет полной. Например:

```
@us.er@
.user@
.user@domain.
.u@d.@
```

Все эти адреса вряд ли можно назвать корректными. Какие есть мысли? Проверять, чтобы в строке был только один символ @. Но все равно есть много неправильных вариантов, например:

```
us.r@domain
пользователь@domain.ru
user@domain...
```

Чтобы не загромождать код лишними проверками, есть один выход - использовать регулярные выражения. Только с помощью регулярных выражений можно быстро и компактно проверить правильность e-mail. Давайте напишем функцию validateEmail(), которая будет проверять правильность введенного e-mail:

```
function validateEmail(email) {
    var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()
[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.
[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]
{2,}))$/;
    return re.test(email);
}
```

Да, регулярное выражение очень сложное, но оно учитывает все возможные варианты правильного построения e-mail и избавляет вас от множества ненужных проверок.

В листинге 11.3 приведен полный код сценария проверки e-mail:

Листинг 11.3. Проверка допустимости e-mail

```
<html>
<head>
<meta charset="utf-8">
<title>Проверка правильности e-mail</title>
<script>

function validateEmail(email) {
var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()
[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.
[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]
{2,}))$/;
return re.test(email);
}
```

```
function check() {

var email = document.form1.email.value;

if (validateEmail(email)) window.alert("OK");
else window.alert("E-mail неправильный");

}

</script>
<body>

 <form name="form1">
 E-mail: <input type="text" name="email" id="email">
 <input type="button" value="Проверить" onclick="check();" >
 </form>

</body>
</html>
```

В функции check() мы передаем содержимое текстового поля email нашей функции validateEmail(). Если она возвращает true, мы выводим сообщение OK, в противном случае сообщаем, что e-mail некорректен.

Теперь давайте усложним задачу. Наверняка вы видели в Интернете сценарии, где проверка ввода осуществляется при самом вводе, а не при нажатии кнопки **Check**. Такие кнопки уже давно в прошлом и в современных сценариях будут выглядеть архаично. Сейчас мы перепишем наш сценарий так, чтобы он проверял правильность e-mail на лету и выводил в определенный элемент документа результат проверки. Первым делом нужно переписать саму функцию check(). Нам уже не нужно выводить сообщения с помощью window.alert(), поэтому результат проверки мы будем выводить в отдельный документа. Вот измененный код:

```
function check() {

var email = document.form1.email.value;
if (validateEmail(email)) document.getElementById("span1").
innerText = "OK";
else  document.getElementById("span1").innerText  =  "Email
недопустим";

}
```

Логика функции осталась той же, изменился только способ отображения результата. Все остальное - дело техники. Для проверки "на лету" нужно использовать событие onKeyDown нашего поля ввода:

```
<input type="text" name="email" id="email"
onKeyDown="check();">
```

В качестве обработчика события мы используем нашу функцию check(). Собственно, в нашей форме уже нет необходимости в кнопке **Проверить**. Эта форма теперь изображена на рис. 11.2. Как видите, она содержит только поле ввода e-mail. Элемент, в который выводится результат обработки, может находиться за пределами формы.

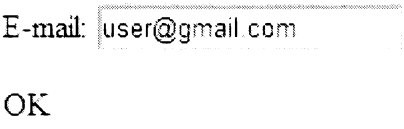


Рис. 11.2. Форма ввода

Измененный сценарий представлен в листинге 11.4.

Листинг 11.4. Измененный сценарий

```
<html>
<head>
<meta charset="utf-8">
<title>Проверка правильности e-mail</title>

<script>

function validateEmail(email) {
    var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()
[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.
[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]
{2,}))$/;
return re.test(email);
}

function check() {

var email = document.form1.email.value;
```

```
if (validateEmail(email)) document.getElementById("span1").
innerText = "OK";
else document.getElementById("span1").innerText = "Email is
not valid";

}

</script>
<body>

 <form name="form1">E-mail: <input type="text" name="email"
id="email"  onKeyDown="check();">
 </form>

<span id="span1"></span>

</body>
</html>
```

Аналогичным образом, используя событие onKeyDown, можно проверить правильность других полей формы, например, указал ли пользователь свой номер телефона. Ниже, кстати, представлена функция для проверки номера телефона в международном формате (+код_страны(код_оператора)xxx-xx-xx):

```
function isValidPhone(Phone) {
          return    /^\+\d{1,2}\(\d{3}\)\d{3}-\d{2}-\d{2}$/.
test(Phone);
}
```

Обратите внимание, что функции проверки e-mail и номера телефона выясняют, соответствуют ли введенные пользователем данные определенному шаблону. Они не проверяют существование такого e-mail и такого номера телефона. Например, вы можете ввести +33(011)010-00-00. Это корректный номер телефона, то есть такой номер может существовать, но существует ли - никто не знает (лично я не проверял). Проверить существование e-mail можно только путем отправки сообщения на этот адрес, но, как правило, этим никто не занимается. С технической точки зрения реализовать такую проверку можно: отправить e-mail, получить ответ от сервера и выдать результат. Однако подобные проверки порождают ненужный трафик.

Также на некоторых сайтах есть возможность проверки не только корректности введенных данных, но и их допустимости. Например, вы при регистрации ввели e-mail user@domain.com. С точки зрения функции проверки он является корректным, поскольку соответствует регулярному выраже-

нию. Однако этот e-mail может принадлежать другому пользователю (кто-то уже указывал его при регистрации). К сожалению, JavaScript не умеет непосредственно обращаться к базе данных, чтобы проверить, есть ли такой e-mail в БД. Поэтому у вас есть два варианта:

- Остановиться на функции validateEmail() и отправлять данные на сервер в случае, если они похожи на правильные. Дальнейшей обработкой будет заниматься программа на сервере (как правило, это будет PHP-сценарий).
- Освоить технологию AJAX (она будет рассмотрена в следующей части книги), позволяющую вызвать стороннюю программу для обработки данных. Например, наш сценарий вызовет PHP-сценарий, который проверит, указывался ли такой e-mail при регистрации другого пользователя, и передаст результат проверки нашему JS-сценарию. Наш сценарий на основании полученного ответа выполнит определенные действия (сообщит пользователю, что e-mail занят или свободен).

11.8. Форма заказа для нашего сайта

Теория - это хорошо, но нам нужна форма заказа для нашего сайта. Первым делом разработаем саму форму. Предлагаю оформить ее как HTML5-диалог, чтобы не ломать себе голову над ее внешним видом. При желании вы можете использовать CSS для оформления формы, мы же этого делать не станем для упрощения примера. Как будет выглядеть наша форма, показано в лист. 11.5 и на рис. 11.3. Пока пусть внешний вид формы вас не тревожит, мы его изменим, когда будем рассматривать создание вкладок с помощью jQuery UI.

Листинг 11.5. Форма заказа

```
 <dialog id="Dialog">
   <p>Форма заказа</p>
   <form name="order" action="http://localhost/order.php"
onsubmit="return validate_form ( );">
   <input type="hidden" name="product" value="">
   <p>Фамилия, имя, отчество: <input type="text" name="fio">
   <p>E-mail: <input type="text" name="email">
   <p>Телефон: <input type="text" name="phone">
   <p>Адрес: <input type="text" name="address">
   <p><input type="submit" value="Отправить">

  </dialog>
```

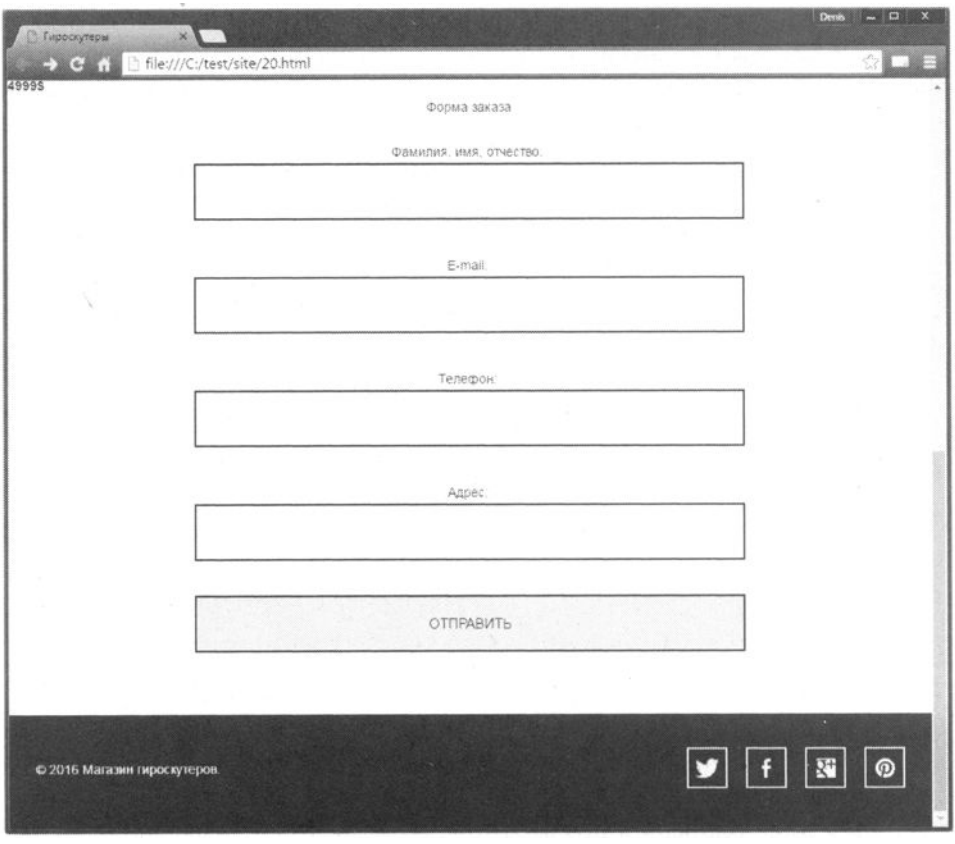


Рис. 11.3. Форма заказа

Разберемся, что есть что. При нажатии кнопки отправки формы (submit) будет вызвана функция проверки формы - validate_form(). Эта функция должна проверить введенные пользователем значения и вернуть одно из двух значений: true или false. Если форма возвращает true, то введенные значения будут переданы сценарию, указанному в action. Если форма вернет false, данные передаваться не будут.

Функция validate_form() должна сообщить пользователю о результатах проверки. Сделать это она может разными способами, например вывести уведомление или же выделить поле с некорректной информацией красным цветом, вывести в документ описание ошибки (например, неправильный e-mail) и т.д. Способы оповещения пользователя зависят только от фантазии программиста.

Код нашей функции проверки приведен в листинге 11.6. Мы проверим только поля Ф.И.О и e-mail. Добавить проверку остальных полей вы сможете по образу и подобию.

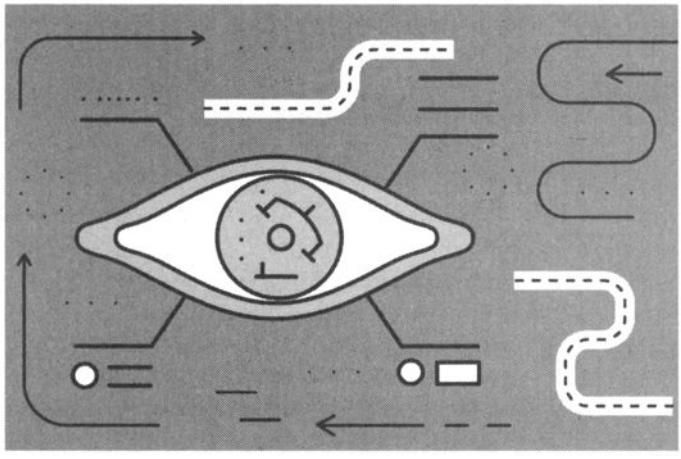
Листинг 11.6. Функция проверки формы

```
function validate_form ( )
{
        valid = true;

        if ( document.order.fio.value == "" )
        {
                alert ("Укажите, пожалуйста, Ф.И.О.");
                valid = false;

        }

        var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()
[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.
[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]
{2,}))$/;

        if (!re.test(document.order.email.value))
        {
                alert ( "Укажите правильный e-mail" );
                valid = false;
        }

        document.order.product.value = window.location.
pathname;
        return valid;
}
```

Обратите внимание: функция проверки не только проверяет поля формы, но и заполняет скрытое поле product. Оно служит для передачи сценарию ссылки на страницу с товаром - так можно будет понять, какой товар заказывал пользователь, и не заполнять его название для каждой страницы. Если вы будете использовать PHP, то можно сразу заполнить это поле при выводе страницы товара.

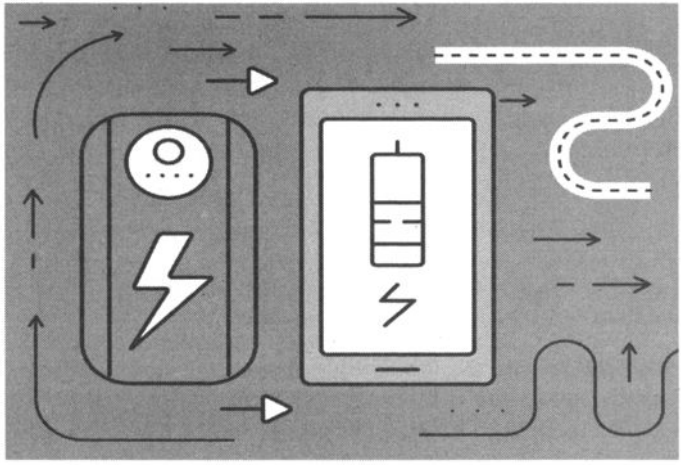
После того как данные будут переданы сценарию order.php, он займется их обработкой, например добавит заказ в базу данных или отправит информацию о нем менеджеру (на e-mail), который потом свяжется с пользователем, сделавшим заказ.


JavaScript™



Глава 12.

Встроенные классы и события JavaScript



Глава 12. Встроенные классы и события JavaScript

В JavaScript имеется богатый набор встроенных (стандартных) классов, содержащих множество полезных методов, которые вы можете использовать при написании своих программ. Если вы пропустили предыдущую главу, настоятельно рекомендую с ней ознакомиться, поскольку она нужна для понимания изложенного в этой главе материала.

12.1. Встроенные классы

12.1.1. Класс Global

Чтобы использовать свойства и методы класса Global вам не нужно создавать экземпляр этого класса. Все методы этого класса доступны как встроенные функции. К свойствам класса относятся NaN (Not a Number, не число) и Infinity (плюс бесконечность). Например:

```
var x = NaN;
var y = Infinity;
```

Методы класса Global:

- parseInt(<строка>, <основание>) - преобразует строку в целое число системы счисления, которая задана вторым параметром. Если второй параметр не указан, по умолчанию используется десятичная система счисления. Если строку нельзя преобразовать в число, функция возвращает значение NaN.
- parseFloat(<строка>) - преобразует строку в число с плавающей точкой.
- eval(<строка>) - вычисляет выражение, заданное в строке, как если бы оно было обычным выражением JavaScript;
- isNaN(<выражение>) - проверяет, является ли выражение правильным числом. Возвращает true, если значение выражения - NaN, false - если значение выражения - обычное число.

- isFinite(<выражение>) - проверяет, является ли выражение конечным числом (возвращает true или false).
- escape(<строка>) - выполняет Escape-кодирование строки (кодирует строку шестнадцатеричными символами).
- unescape(<строка>) - обратное преобразование строки.
- encodeURI(<строка>) - выполняет URI-кодирование строки (полезно при формировании URI, содержащего символы национальных алфавитов).
- decodeURI(<строка>) - обратное кодирование строки.

Рассмотрим несколько примеров:

```
var str = "100";
var x = 50 + parseInt(str);            // 150
var strf = "100.52";
var y = strFloat(strf);               //100.52
var z = eval("2 + 2");                // 4
```

12.1.2. Класс Number

Класс Number используется для работы с числами. Экземпляр класса можно создать так:

```
var <объект> = new Number(начальное значение);
```

Например:

```
var x = Number(1);
```

Свойства класса Number (можно использовать без создания экземпляра класса):

- MAX_VALUE - максимально допустимое в JS число.
- MIN_VALUE - минимально допустимое в JS число.
- NaN - значение NaN.
- NEGATIVE_INFINITY - "минус бесконечность".
- POSITIVE_INFINITY - положительная бесконечность.

К методам этого класса относят всего два метода:

- valueOf() - возвращает числовое значение экземпляра класса.
- toString() - возвращает строковое представление числа.

Примеры:

```
var x = Number.MAX_VALUE;
var y = new Number(100);
var str = y.toString();        // "100"
```

12.1.3. Класс String

Класс String используется для обработки строк. Экземпляр класса можно создать так:

```
var <объект> = new String (<строка>);
```

Например:

```
var so = new String ("Привет");
```

Но мы уже знаем, что строку можно создать гораздо проще, например:

```
var s = "Hello";
```

Разница в том, что тип переменной s будет string, а переменной so - object. Однако к обычным строкам можно применять методы класса String, например:

```
var str = "Привет, мир!".toUpperCase();
```

При использовании метода toUpperCase() строка, имеющая тип данных string, будет автоматически преобразована в объект класса String. Класс String является оберткой над типом данных string.

У класса String есть всего одно свойство - lenght, содержащее длину строки:

```
var so = new String ("Привет");
document.write(so.length);                    // 5
```

А вот методов у класса String значительно больше, поэтому они приведены в таблице 12.1 для большего удобства чтения.

Таблица 12.1. Методы класса String

Метод	Описание
toString()	Преобразует объект класса String в строку
valueOf()	Возвращает значение хранящейся в объекте строки

charAt(<номер символа>)	Возвращает символ строки с указанным номером. Нумерация символов начинается с 0. Например:
	`var s = "JavaScript";`
	`var x = s.charAt(0);  // "J"`

charCodeAt	Возвращает код символа строки с указанным номером. Нумерация начинается с 0. Пример:
	`var s = "Hello";`
	`var x = s.charCodeAt(0);  // 72`

fromCharCode (<код1>, ..., <кодN>)	Создает строку из указанных кодов

toLowerCase()	Преобразует символы строки в нижний регистр. Пример:
	`var s = "JavaScript";`
	`s = s.toLowerCase(); // javascript`

toUpperCase()	Преобразует символы строки в символы верхнего регистра

substr(<начало фрагмента>, [длина])	Извлекает фрагмент строки заданной длины. Если второй параметр пропущен, будут возвращены все символы до конца строки.
	`var S = "Hello, world";`
	`window.alert(S.substr(0, 5)); // "Hello"`
	`window.alert(S.substr(7)); // "world`

substring(<начало фрагмента>,<конец фрагмента>)	Извлекает фрагмент строки, заданный номерами начального и конечного символов. Последний символ в подстроку не включается, например:
	`var S = "Hello, world";`
	`document.write(S.substring(7, 12)); // "world"`

indexOf(<подстрока>, [<начальная позиция поиска>])	Возвращает номер позиции первого вхождения подстроки в текущей строке. Если подстрока не найдена, возвращает -1. Пример:
	`var S = "Hello, world";`
	`document.write(S.indexOf("llo")); // 2`

lastIndexOf(<подстрока>, [<начальная позиция поиска>])	Возвращает номер позиции последнего вхождения подстроки в текущей строке. Если подстрока не найдена, возвращает -1.

split(<Разделитель>, [<Лимит>])	Разделяет строку на подстроки по символу-разделителю и возвращает массив. Например:
	`var S = "Hello, world";`
	`var M = S.split(",");`
	`window.alert(Mass[0]); // "Hello"`
	`window.alert(Mass[1]); // " world"`

search(<регулярное выражение>)	Определяет номер позиции первого вхождения подстроки, совпадающей с регулярным выражением

match(<регулярное выражение>)	Возвращает массив с результатами поиска, которые совпадают с регулярным выражением
replace(<регулярное выражение>, <текст для замены>)	Выполняет поиск и замену в исходной строке с использованием регулярного выражения.

12.1.4. Класс Array

Свойства и методы класса

Ранее мы говорили о создании массивов. Сейчас мы рассмотрим класс Array, который можно использовать для обработки массивов, а также для их создания. Создать экземпляр этого класса можно так:

```
var <объект> = new Array (<к-во элементов массива>);
var <объект> = new Array (<элементы массива через запятую>);
```

Пример:

```
var M = new Array(1, 2, 3);
```

Свойство length содержит количество элементов массива. Нумерация элементов массива начинается с 0. Пример использования свойства length:

```
document.write(M.length);

for (var i=0, c=M.length; i<c; i++) {
document.write(M[i] + "<br>");
}
```

Методов у этого класса не меньше, чем у класса String. Онипривedены в таблице 12.2.

Таблица 12.2. Методы класса Array

Метод	Описание
push(<элементы>)	Добавляет элементы в конец массива, возвращает новую длину массива: `M.push(4, 5);`

unshift(<элементы>)	Добавляет элементы в начало массива

concat(<элементы>)	Возвращает массив, полученный в результате объединения текущего массива и списка элементов. В текущий массив элементы из списка не добавляются. Например:
	`var M1 = new Array(1, 2, 3);`
	`var M2 = [];`
	`M2 = M.concat(4, 5);`
	`// M1 = [1, 2, 3]`
	`// M2 = [1, 2, 3, 4, 5]`

join(<разделитель>)	Объединяет элементы массива в строку, разделяя их заданным разделителем. Пример:
	`var M1 = new Array(1, 2, 3);`
	`document.write(M1.join(" "));  // 1 2 3`

shift()	Возвращает первый элемент массива и удаляет его из массива
pop()	Возвращает последний элемент массива и удаляет его из массива
sort(<функция сортировки>)	Выполняет сортировку массива. Если функция сортировки не указана, будет выполнена обычная сортировка (числа - по возрастанию, символы - по алфавиту).
reverse()	Переворачивает массив. Элементы массива будут в обратном порядке.

slice(<начало>, [конец])	Возвращает часть массива, начиная от индекса <начало> до индекса <конец>. Если второй индекс не задан - до конца массива. Пример:
	`var M1 = new Array(1, 2, 3, 4, 5, 6);`
	`var M2 = M1.slice(1, 3);  // [2, 3]`

toString()	Преобразует массив в строку. Элементы указываются через запятую без пробела.

Сортировка массива

Отдельного внимания заслуживает сортировка массива. Функция sort() способна выполнить только базовую сортировку массива, которой не всегда достаточно, поэтому программист может создать собственную функцию сортировки. Такая функция должна принимать две переменные и возвращать:

- 0 — если обе переменные равны;
- -1 — если вторая переменная больше первой;
- 1 — если первая переменная больше второй.

Рассмотрим пример сортировки строк без учета регистра:

```
function f_sort(Str1, Str2) {
   var S1 = Str1.toLowerCase(); // Преобразуем к нижнему
регистру
   var S2 = Str2.toLowerCase(); // Преобразуем к нижнему
регистру
   if (S1>S2) return 1;
   if (S1<S2) return -1;
   return 0;
}

var Mass = [ "One", "two9", "open" ];
Mass.sort(f_sort);
document.write(Mass.join(", "));
```

Порядок сортировки можно изменить, поменяв возвращаемые значения на противоположные:

```
function f_sort(Str1, Str2) {
   var S1 = Str1.toLowerCase(); // Преобразуем к нижнему
регистру
   var S2 = Str2.toLowerCase(); // Преобразуем к нижнему
регистру
   if (S1>S2) return -1;
   if (S1<S2) return 1;
   return 0;
}
```

Многомерные массивы

Многомерные массивы можно создать несколькими способами. Например, можно их создавать поэлементно:

```
var M = [];
M[0] = [];
M[1] = [];
M[0][0] = 1;
M[0][1] = 2;
M[0][2] = 3;
M[1][0] = 3;
M[1][1] = 2;
M[1][2] = 3;
```

Также можно использовать перечисление, например:

```
var M = new Array(new Array("1", "2", "3"),
                  new Array("3", "2", "1"));
```

Ассоциативные массивы

Если вы программировали на PHP, то наверняка знакомы с ассоциативными массивами. Ассоциативные массивы позволяют в качестве индексов использовать строки, а не только числа. Пример:

```
var M = new Array();
M ["one"] = 1;
M ["two"] = 2;
```

Ни один из методов класса Array не позволяет вывести элементы ассоциативного массива. Свойство length также не работает, поэтому вы не можете перебрать элементы ассоциативного массива в цикле for. Для этого нужно использовать цикл for ..in:

```
for (var N in M) {
      document.write(N + " = " + M[N] + "<br>");
}
```

Вывод будет таким:

```
one = 1
two = 2
```

12.1.5. Класс Math

Класс Math содержит некоторые математические функции и константы. Его использование не требует создания экземпляра класса.

В классе Math содержатся следующие константы:

- E – экспонента, основание натурального логарифма.
- LN2 – натуральный логарифм 2.
- LN10 – натуральный логарифм 10.
- LOG2E – логарифм по основанию 2 от E.
- LOG10E – логарифм по основанию 10 от E;
- PI – число Пи.
- SQRT2 – квадратный корень из 2.
- SQRT1_2 – квадратный корень из 0,5.

Методы класса Math представлены в таблице 12.3.

Таблица 12.3. Методы класса Math

Метод	Описание
abs()	Возвращает абсолютное значение
sin(), cos(), tan(), asin(), acos(), atan()	Стандартные тригонометрические функции
exp()	Экспонента
log()	Натуральный логарифм
pow(<число>, <степень>)	Возведение в степень
sqrt()	Квадратный корень

round()	Значение, округленное до ближайшего целого. Округление может быть, как в большую, так и меньшую сторону
ceil()	Значение, округленное до ближайшего большего целого.
floor()	Округление до ближайшего меньшего целого.
max(<список элементов через запятую>), min(<список элементов через запятую>)	Возвращают максимальное/минимальное значение из списка
random()	Возвращает случайное число от 0 до 1.

Как использовать математические функции на практике? Пусть у нас есть 4 баннера и нужно их выводить случайным образом при обновлении страницы. Вот пример кода:

```
var n = Math.floor(Math.random()*3.9999);
document.write('<img src="banner' + n + '.gif">');
```

Файлы с баннерами должны называться banner0.gif...banner3.gif.

12.1.6. Классы Function и Arguments

Класс Function позволяет использовать функцию в качестве экземпляра класса:

```
<Имя функции> = new Function(<аргумент 1>, …, <аргумент N>,
<код>);
```

Пример:

```
var Sum = new Function("x", "y", "return x+y");
```

Однако таким способом мало кто пользуется, поскольку указывать код функции в виде строки очень неудобно. Зато можно использовать анонимные функции, например:

```
var Sum = function(x, y) { return x + y; }
```

Вызвать функцию можно, как и раньше:

```
window.alert(Sum(2,2));
```

В JavaScript можно создавать функции с произвольным числом аргументов. Доступ ко всем указанным при вызове функции аргументам осуществляется через массив arguments, который доступен только внутри тела функции. Свойство length этого массива содержит число переданных аргументов. Напишем функцию Sum, вычисляющую сумму произвольного числа аргументов:

```
function Sum() {
      var r = 0;
      for (var i=0; i < arguments.length; i++) r = r +
arguments[i];
      return r;
}
window.alert(Sum(1, 2, 3));
```

12.1.7. Класс Date

Для работы с датой и временем в JavaScript используется класс Date. Есть несколько способов создать экземпляры этого класса:

```
var <объект> = new Date();
var <объект> = new Date(<количество миллисекунд>);
var <объект> = new Date(<год>, <месяц>, <день>, <часы>,
<мин>, <с>, <мс>);
```

Класс Date содержит много методов, которые вместе с примерами по их использованию, приведены в таблице 12.4.

Таблица 12.4. Методы класса Date

Метод	Описание

toString()	Преобразует дату в строку и возвращает ее.
	Пример:
	`var d = new Date(); // текущая дата`
	`document.write(d.toString());`
	Вывод может отличаться в зависимости от браузера. В Chrome вывод будет таким:
	`Thu Jan 30 2014 15:37:14 GMT+0200 (Финлян-дия (зима))`

toLocaleString()	Преобразует дату в строку с использованием интернациональных установок системы. Параметры моей системы таковы, что Chrome отобразил строку: `30.1.2014 15:38:57`

valueOf()	Возвращает число секунд, прошедших с 01.01.1970 00:00:00.
	`var d = new Date(); // текущая дата`
	`document.write(d.valueOf());`
	Вывод:
	`1391089258015`

getDate()	Возвращает день месяца (от 1 до 31)
getDay()	Возвращает день недели, 0 - воскресение, 1 - понедельник и т.д.
getMonth()	Возвращает номер месяца (0 - январь, 11 - декабрь)

getFullYear()	Возвращает полный год (например, 2014)
gertHours()	Возвращает час (от 0 до 23)
getMinutes()	Возвращает минуты (от 0 до 59)
getSeconds()	Возвращает секунды (от 0 до 59)
getMilliseconds()	Возвращает миллисекунды (от 0 до 999)
getTime()	Возвращает то же значение, что и valueOf()

Напишем небольшую функцию, возвращающую название месяца по переданному ей номеру месяца:

```
function getStrMonth(m) {

  var d = new Date();

  var Months = [ "Jan", "Feb", "Mar", "Apr", "May",
  "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ];

  var currentMonth = d.getMonth();

  if ((m < 0) || (m > 11)) return Months[CurrentMonth];

   return Months[m];    // нумерация массива с 0, нумерация
месяцев с 0
}

var d = new Date();
window.alert(getStrMonth(d.getMonth()));
```

В нашей функции определен массив Months, содержащий символьные названия (в сокращенном виде) месяцев. Затем функция определяет текущий месяц, вызвав getMonth() - на случай, если она будет вызвана с неправильным номером месяца (который меньше 0 или больше 11). В этом случае функция вернет название текущего месяца (а не сообщение об ошибке - сделаем нашу функцию более дружественной). Это позволяет вместо лишнего вызова getMonth() в программе просто указать неподходящий номер месяца, например -1, и вы получите название текущего месяца, а не сообщение об ошибке. Например, следующий код в программе:

```
var d = new Date();
window.alert(getStrMonth(d.getMonth()));
```

можно смело заменить на такой код:

```
window.alert(getStrMonth(-1));
```

Так мы сокращаем число строк в программе. Нашу функцию можно использовать как в паре с методом getMonth (что было продемонстрировано выше), так и напрямую указывать номер месяца (только помните, что нумерация месяцев начинается с 0!).

12.1.8. Класс RegExp

Класс RegExp используется для работы с регулярными выражениями, а именно для организации поиска в строке с помощью регулярных выражений. Регулярные выражения - это шаблоны для поиска определенных комбинаций метасимволов и позволяют осуществить очень сложный поиск. Экземпляр класса RegExp создает так:

```
var <объект> = new RegExp(<регулярное выражение>,
<модификатор>);
var <объект> = /<регулярное выражение>/[<модификатор>];
```

Модификатор может принимать следующие значения:

- i — поиск без учета регистра;
- g — глобальный поиск;
- m — многострочный поиск;
- gi — глобальный поиск без учета регистра.

Подробное рассмотрение регулярных выражений, к сожалению, выходит за рамки этой книги. Стоит заметить, что регулярные выражения, поддерживаемые в JavaScript, используются и в других языках программирования, например, в PHP, что добавит вашему коду больше портабельности. Подробно о регулярных выражениях в JavaScript вы можете прочитать по адресу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

Далее мы рассмотрим несколько примеров, демонстрирующих всю силу регулярных выражений. При рассмотрении класса String вы узнали, что есть

три метода, работающие с регулярными выражениями: search(), match() и replace(). Метод search() возвращает номер позиции первого вхождения подстроки, которая совпадает с регулярным выражением:

```
var p = new RegExp("abc[de]");
var Str = "abc, abcd, abce, abcf";
document.write(Str.search(p));              // 5
```

В результате будет выведено 5, поскольку первая подстрока, соответствующая регулярному выражению "abc[de]", начинается с позиции 5 (шестой символ, нумерация с 0). Вообще в строке Str есть две подстроки, которые соответствуют нашему регулярному выражению, - abcd и abce, но функция возвращает позицию только первой найденной подстроки.

Если необходимо найти все соответствия, тогда нужно использовать метод match(), который возвращает массив с результатами поиска:

```
var p = new RegExp("abc[de]");
var Str = "abc, abcd, abce, abcf";
var M = [];

M = Str.match(p);

for (var i=0, c=M.length; i<c; i++)
  document.write(M[i] + " ");
```

Этот пример выведет только abcd, чтобы match() вывел остальные варианты, нужно использовать модификатор глобального поиска:

```
var p = new RegExp("abc[de]", "g");
```

Вот тогда результат будет таким, как нужно:

```
abcd abce
```

Метод replace(<регулярное выражение>, <текст для замены>) используется для поиска и замены текста с использованием регулярного выражения.

```
var p = new RegExp("abc[de]", "g");
```

```
var Str = "abc, abcd, abce, abcf";
var S = Str.replace(p, "match");
document.write(S);
```

В результате будет выведена строка:

```
abc, match, match, abcf
```

Все найденные подстроки, соответствующие регулярному выражению, будут заменены на "match".

На этом мы заканчиваем рассмотрение встроенных классов JavaScript и переходим к рассмотрению событий.

12.2. События JavaScript

12.2.1. Что такое событие?

События происходит при взаимодействии пользователя с веб-страницей. Посредством событий система извещает сценарий, что пользователь выполнил какое-то действие: переместил мышь, нажал кнопку мыши, нажал какие-то клавиши на клавиатуре, изменил размеры окна и т.д. Например, при нажатии кнопки мыши возникает событие onmousedown. Названия всех событий начинаются с префикса on.

12.2.2. События мыши

К событиям мыши относятся события, описанные в таблице 12.5.

Таблица 12.5. События мыши

Событие	Когда происходит
onmousedown	Происходит при нажатии кнопки мыши на элементе веб-страницы или самой странице.
onmouseup	Когда пользователь отпускает ранее нажатую кнопку мыши.

Onclick	Происходит, когда пользователь щелкает на элементе веб-страницы или самой странице
Ondblclick	При двойном щелчке
Onmouseover	Когда пользователь наводит курсор мыши на элемент страницы (картинку, надпись, абзац, кнопку и т.д.)
Onmouseout	Когда курсор мыши выходит за пределы элемента страницы
Onmousemove	При перемещении мыши (при любом перемещении)
Onselect	При выделении элемента
Onselectstart	При начале выделения
Oncontextmenu	При выводе контекстного меню (когда пользователь нажимает правую кнопку или левую кнопку, если пользователь - левша).

12.2.3. События клавиатуры

Аналогично, события клавиатуры описаны в таблице 12.6.

Таблица 12.6. События клавиатуры

Событие	Описание
Onkeydown	При нажатии клавиши на клавиатуре
Onkeypress	Тоже при нажатии клавиши на клавиатуре, но возвращает код нажатого символа в Unicode
Onkeyup	Когда пользователь отпускает нажатую ранее клавишу
Onhelp	При нажатии клавиши F1

12.2.4. События документа

В таблице 12.7 приведены основные события документа.

Таблица 12.7. События документа

Событие	Описание
Onload	Происходит сразу после загрузки веб-страницы
Onscroll	При прокручивании содержимого страницы
Onresize	При изменении размеров окна
Onunload	При выгрузке документа. Наступает после события onbeforeunload
Onbeforeunload	Перед выгрузкой документа, перед событием onunload
Onbeforeprint	Перед распечаткой документа
Onafterprint	После распечатки документа

12.2.5. События формы

События формы представлены в таблице 12.8.

Таблица 12.8. События формы

Событие	Описание
Onsubmit	При отправке формы (при нажатии кнопки Submit)
Onreset	При сбросе формы (при нажатии кнопки Reset)
Onblur	Когда элемент формы (например, поле ввода) теряет фокус
Onfocus	Когда элемент формы (например, поле ввода или кнопка) получает фокус (становится активным)
Onchange	Когда фокус перемещается на другой элемент кнопки (наступает перед onblur)

12.2.6. Последовательность событий

События возникают в определенной последовательности, например при нажатии кнопки мыши последовательность будет такой: onmousedown,

onmouseup, onclick. При двойном нажатии мыши последовательность будет немного другой: onmousedown, onmouseup, onclick, ondblclick.

Последовательность событий нужно учитывать при установке обработчиков событий - в зависимости от желаемого результата.

Рассмотрим сценарий (листинг 12.1), демонстрирующий последовательность событий. В нем мы устанавливаем обработчики событий (подробно о них мы поговорим в разд. 12.2.9), чтобы проследить их последовательность.

Листинг 12.1. Пример обработки событий мыши

```
<html>
<head>
<title>События</title>
</head>
<body onload="window.alert('OnLoad. Нажмите кнопку мыши');"
 onmousedown="document.write('OnMouseDown');"
 onmouseup="document.write('OnMouseUp');"
 onclick="document.write('OnClick');">
 <h1>Пример обработки событий мыши</h1>
</body>
</html>
```

Прежде чем перейти к следующему разделу, нужно сделать несколько замечаний. Во-первых, описанная выше последовательность событий мыши верна только для браузера Internet Explorer. В других браузерах последовательность событий и вообще логика работы механизма событий может отличаться. Во-вторых, на практике события onmousedown и onmouseup используются крайне редко, может, в каких-то игровых сценариях. В основном используется событие onclick, когда пользователь щелкнул на элементе веб-страницы. Данное событие можно определить отдельно для элемента, например для изображения:

```
<img src="button.png" onclick="window.alert('OnClick');">
```

12.2.7. Всплывание событий

При работе с событиями приходится иметь дело с явлением, которое называется всплыванием событий. Чтобы понять, что это такое, давайте рассмотрим листинг 12.2.

Листинг 12.2. Всплывание событий

```
<html>
<head>
<title>События</title>
</head>
<body onclick="window.alert('OnClick для документа');">
  <p onclick="window.alert('OnClick для абзаца');">Щелкните
    <span style="color: green" onclick="window.alert('OnClick
для Span');">здесь</span>
  </p>
</body>
</html>
```

После загрузки страницы вы увидите надпись:

```
Щелкните здесь
```

При щелчке на "here" возникнет целая цепочка событий:

```
OnClick для Span
OnClick для абзаца
OnClick для документа
```

Получается, что событие onclick передается последовательно родительскому элементу. Это явление и называется всплыванием событий. Честно говоря, оно не всегда желательно. Поэтому в JavaScript предусмотрена возможность прерывания всплывания событий. Для этого нужно присвоить свойству cancelBubble объекта event значение true. В некоторых браузерах для прерывания всплывания событий используется метод stoPropogation(). Рекомендуется использовать оба способа прерывания одновременно для совместимости с большим числом браузеров.

Пример прерывания всплывания событий приведен в листинге 12.3. Чтобы присвоить свойству cancelBubble значение true, нам пришлось переписать обработчик события. Теперь в качестве обработчика используется наша функция f_alert(), которая устанавливает необходимое свойство и выводит переданное ей сообщение.

Листинг 12.3. Пример прерывания всплывания событий

```
<html>
<head>
```

```
<title>События</title>

<script>
function f_alert(s, e) {
e = e || window.event;
if (e.stopPropogation) e.stopPropogation();
else e.cancelBubble = true;
window.alert(s);
}
</script>

</head>
<body onclick="f_alert('OnClick для документа', event);">
  <p onclick="f_alert('OnClick для абзаца', event);">Щелкните
    <span style="color: green" onclick="f_alert('OnClick для
Span', event);">здесь</span>
  </p>
</body>
</html>
```

В итоге, когда вы щелкните на надписи "здесь", вы получите только одно событие:

```
OnClick для Span
```

12.2.8. Действие по умолчанию

Для некоторых событий назначены действия по умолчанию, например при нажатии кнопки Submit формы идет отправка содержимого формы на веб-сервер сценарию, который задан в свойствах формы. В некоторых ситуациях действия по умолчанию нужно отменить. Для этого нужно установить свойство returnValue в false или же использовать метод preventDefault (поддерживается не всеми браузерами):

```
function cancel(e) {
      e = e || window.event;
      if (e.preventDefault) e.preventDefault();
            else e.returnValue = false;
}
```

Далее для элемента, для которого нужно прервать действие по умолчанию, нужно вызвать эту функцию, например,

```
<img src="button.png" onclick="cancel(event);">
```

12.2.9. Обработчики событий

Как устанавливать обработчики событий, вы уже знаете. Для этого нужно задать событие и указать JavaScript-код:

```
<тег событие="код">
```

Примеров было приведено уже предостаточно. Учитывая, что на практике не всегда нужно выполнять простые действия, заключающиеся из одного-двух операторов (например, вызов window.alert), рекомендуется создавать функции-обработчики событий, что мы и делали в этой главе. Этим вы убиваете двух зайцев: делаете ваш код проще для восприятий (а отсюда повышается читабельность, отлаживаемость кода и упрощается поиск ошибок) и позволяете преодолеть ограничение на максимальную длину значения HTML-атрибута (1024 символа).

Функции-обработчики желательно определять в разделе HEAD - так будет логичнее для вас самих. Если же вы поместите код в BODY, - не беда, браузер правильно обработает JS-код, но это будет не так нагляднее, как если бы обработчики были определены в HEAD. Например, в следующем сценарии (лист. 12.4) обработчик onload все равно будет вызван, хотя на самом деле он определен после тега body. Это происходит потому, что браузер сначала обрабатывает код сценария, а потом уже HTML-код.

Листинг 12.4. Пример обработчика события

```
<html>
<head>
  <title>Events</title>
</head>

<body onload="msg();">
<script>
function msg() {
window.alert('Привет'); }
</script>
</body>

</html>
```

12.2.10. Объект event

В этой главе мы уже использовали объект event. Однако он заслуживает отдельного разговора, и мы его рассмотрим именно сейчас. Объект event используется для получения подробной информации о произошедшем событии и доступен только в обработчиках событий (в других функциях и методах вы не можете его использовать). При наступлении следующего события все значения свойства объекта event сбрасываются. Свойства объекта event приведены в таблице 12.9.

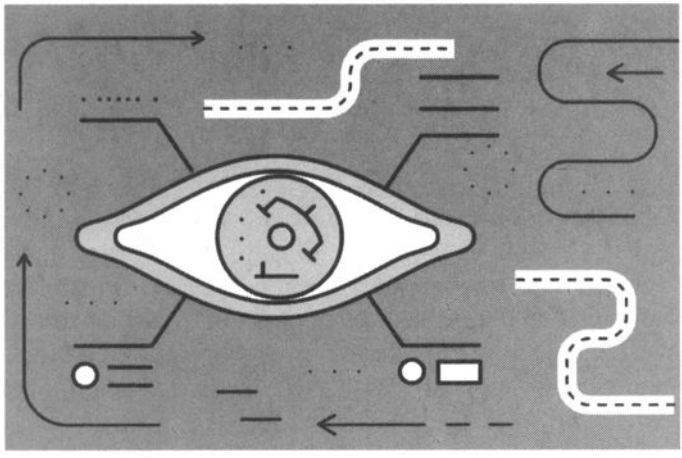
Таблица 12.9. Свойства объекта event

Свойство	Описание
srcElement	Ссылка на элемент, который является источником события.
currentTarget	Возвращает ссылку на элемент, в котором обрабатывается событие. Ссылается на тот же элемент, что и ключевое слово this внутри обработчика события.
Type	Содержит тип события (строку). Возвращается в нижнем регистре и без префикса on, например, для onmousedown свойство type равно "mousedown"
clientX, clientY	Координаты (X, Y) события
screenX, screenY	Координаты (X, Y) события относительно окна
offsetX, offsetY	Координаты (X, Y) события относительно контейнера
button	Число, указывающее, какая кнопка мыши была нажата: 0 - левая, 1 - средняя, 2 - правая
keyCode	Код нажатой на клавиатуре клавиши. В некоторых браузерах, например в Firefox, это свойство при обработке события onkeypress равно 0, а код символа доступен через свойство charCode. Если нажата функциональная клавиша, тогда charCode = 0, а код символа находится в keyCode.

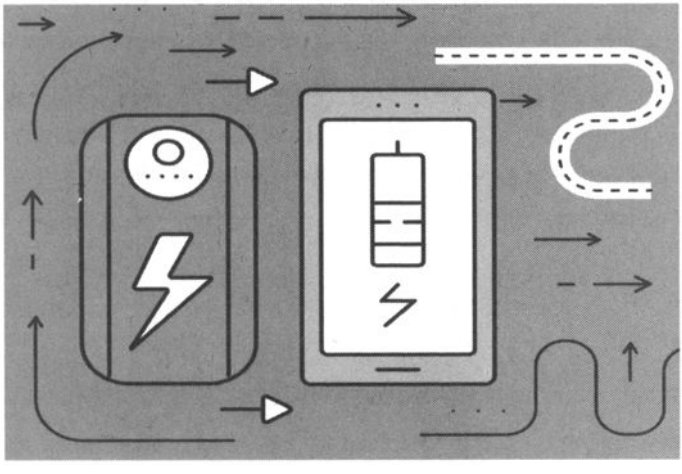
altKey	Если это свойство равно true, то была нажата клавиша Alt вместе с другой клавишей, например пользователь мог нажать Alt + A
ctrlKey	Если это свойство равно true, то была нажата клавиша Ctrl
shiftKey	Если это сойство равно true, то была нажата Shift
cancelBuble	Запрещает всплывание событий. Использование этого свойства было продемонстрировано ранее в этой главе.
returnValue	Определяет, будет ли выполняться действие по умолчанию. Использование этого свойства было продемонстрировано ранее в этой главе.
relatedTarget	Ссылка на элемент, с которого перешел курсор мыши.

В таблице 12.9 описаны не все возможные свойства объекта event. Наличие или отсутствие конкретного свойства зависит от уровня DOM. DOM (Document Object Model) - объектная модель документа - это независимый от платформы и языка программирования интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление этих документов. Существует четыре уровня DOM: DOM Level 0, DOM Level 1, DOM Level 2, DOM Level 3.

Стандарты DOM Level 0 и Level 1 так устарели, что заглядывать в них нет смысла. На смену Level 1 пришел Level 2, который внес много изменений в первые два уровня. В таблице 15.5 приведены свойства event согласно стандарта DOM Level 2. Также есть и самый новый стандарт Level 3, но он используется неохотно, несмотря на то, что был принят уже довольно давно - в 2009 году.



Глава 13.
Эффектная лендинг-страница



В этой главе будет показано, как сделать эффектную лендинг-страницу для нашего сайта. Данная страница используется для привлечения аудитории и, как следствие этого, увеличения продаж. Некоторые сайты в Интернете состоят всего из одной лендинг-страницы. Такие страницы в основном используются, когда товара мало - от одного до пяти наименований. Для полноценного интернет-магазина возможностей лендинг-страницы будет мало.

За основу для нашей лендинг-страницы мы возьмем официальный сайт Nike http://www.nikebetterworld.com/ и попытаемся реализовать подобный эффект. Зайдите на сайт Nike и прокрутите колесиком страницу вниз. Именно такую лендинг-страницу мы создадим для нашего сайта по продажам гироскутеров.

13.1. Необходимые сценарии

Для создания подобного эффекта перелистывания нам понадобятся следующие сценарии:

- jQuery 1.4.4.
- jQuery Parallax - собственно он и организует перелистывание.
- jQuery localscroll - позволяет организовать плавную прокрутку между статьями.
- jQuery scrollTo - также нужен для плавной прокрутки.
- jQuery Inview - определяет, какая статья просматривается.

Не волнуйтесь: далее будет приведена ссылка, по которой можно будет скачать рабочий пример со всеми этими сценариями.

Все эти сценарии подключаются в секции <head>:

```
<script type="text/javascript" src="https://ajax.googleapis.
com/ajax/libs/jquery/1.6.4/jquery.min.js"></script>
```

```
<script type="text/javascript" src="scripts/jquery.parallax-
1.1.3.js"></script>
<script type="text/javascript" src="scripts/jquery.
localscroll-1.2.7-min.js"></script>
<script type="text/javascript" src="scripts/jquery.scrollTo-
1.4.2-min.js"></script>
```

Для ускорения загрузки страницы я рекомендую скачать файл https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js на собственный сервер и вызывать его локально.

Вспомогательный сценарий определяет слайды со статьями:

```
<script type="text/javascript">
$(document).ready(function(){
      $('#nav').localScroll(800);
      $('#intro').parallax("50%", 0.1);
      $('#second').parallax("50%", 0.1);
      $('.bg').parallax("50%", 0.4);
      $('#third').parallax("50%", 0.3);

})
</script>
```

Первый параметр, передающийся в parallax(), - это горизонтальное положение элемента, второй - скорость вертикальной прокрутки. При этом 0.1 - это стандартная скорость, 0.2 - скорость в два раза выше и т.д. Третий параметр необязательный, и он позволяет задать, будет ли (true) или нет (false) библиотека jQuery использовать свою опцию outerHeight для определения секции в viewport. Этот параметр можно не задавать.

13.2. HTML-код

Навигация определяется так:

```
<ul id="nav">
  <li><a href="#intro" title="1"><img src="images/dot.png"
alt="" /></a></li>
  <li><a href="#second" title="2"><img src="images/dot.png"
alt="" ></a></li>
```

```
  <li><a href="#third" title="3"><img src="images/dot.png"
alt="" /></a></li>
  <li><a href="#fifth" title="4"><img src="images/dot.png"
alt="" /></a></li>
</ul>
```

Как видите, мы задаем id блоков div для каждой статьи, а также изображения, которые будут использоваться для доступа к тому или иному блоку. Мы используем изображение точки для доступа ко всем блокам.

Затем нужно описать сами статьи, вот пример статьи intro:

```
      <div id="intro">
            <div class="story">
            <div class="float-left">
            <h2>Дешевле, чем смартфон</h2>
              <p>Стоит всего 300$, запас хода от 20 км,
скорость от 10 км/ч</p>
              </div>
          </div> <!--.story-->
      </div> <!--#intro-->
```

Остальные статьи описываются аналогично. Теперь заглянем в style.css. В нем описываются стили. Больше всего нас интересуют изображения и их параметры:

```
#intro{
      background:url(images/1.jpg) 50% 0 no-repeat fixed;
      color: white;
      height: 700px;
      margin: 0 auto;
      padding: 0;
}

#second{
      background: url(images/2.jpg) 50% 0 no-repeat fixed;
      color: black;
      height: 1200px;
      margin: 0 auto;
      overflow: hidden;
      padding: 0;
}
```

Думаю, понятно, где прописываются сами изображения. Параметр height задает высоту изображения. Для правильной работы сценария вам нужно отредактировать эти значения, если вы замените изображения другими. Это значение должно быть немного меньше, чем реальная высота изображения. Параметр color задает цвет текста статьи. Установите такой цвет, чтобы его было нормально видно на фоне изображения.

13.3. Сценарий прокрутки

Сценарий, обеспечивающий прокрутку изображений (статей) приведен в листинге 13.1.

Листинг 13.1. Сценарий jquery-parallax

```
(function( $ ){
   var $window = $(window);
   var windowHeight = $window.height();

   $window.resize(function () {
      windowHeight = $window.height();
   });

   $.fn.parallax = function(xpos, speedFactor, outerHeight) {
      var $this = $(this);
      var getHeight;
      var firstTop;
      var paddingTop = 0;

      // Получаем начальную позицию каждого элемента

      $this.each(function(){
          firstTop = $this.offset().top;
      });

      if (outerHeight) {
         getHeight = function(jqo) {
            return jqo.outerHeight(true);
         };
      } else {
         getHeight = function(jqo) {
            return jqo.height();
         };
```

```
    }

    // Устанавливаем значения по умолчанию, если аргументы не
указаны
    if (arguments.length < 1 || xpos === null) xpos = "50%";
    if (arguments.length < 2 || speedFactor === null)
speedFactor = 0.1;
    if (arguments.length < 3 || outerHeight === null)
outerHeight = true;

    // Эта функция будет вызвана, если окно будет прокручено
или
     // изменен его размер
    function update(){
      var pos = $window.scrollTop();

      $this.each(function(){
        var $element = $(this);
        var top = $element.offset().top;
```

Рис. 13.1. Начальная позиция

```
            var height = getHeight($element);

            // Помещаемся ли мы во viewport
            if (top + height < pos || top > pos + windowHeight)
{
               return;
            }

            $this.css('backgroundPosition', xpos + " " + Math.
round((firstTop - pos) * speedFactor) + "px");
         });
      }

      $window.bind('scroll', update).resize(update);
      update();
   };
})(jQuery);
```

Результат приведен на рис. 13.1 и 13.2.

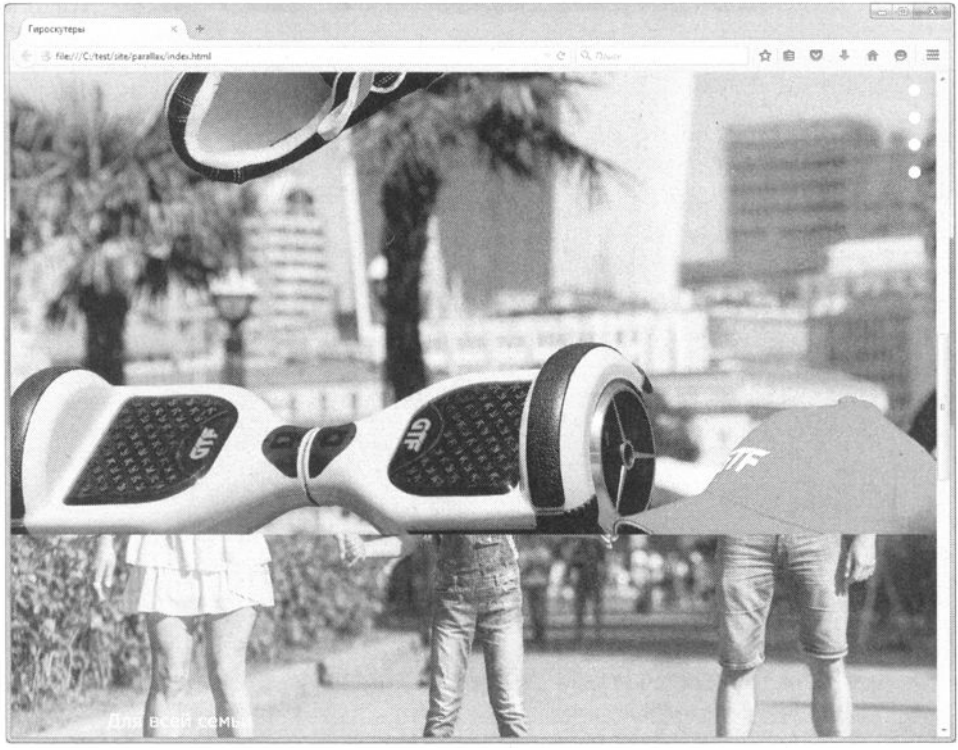
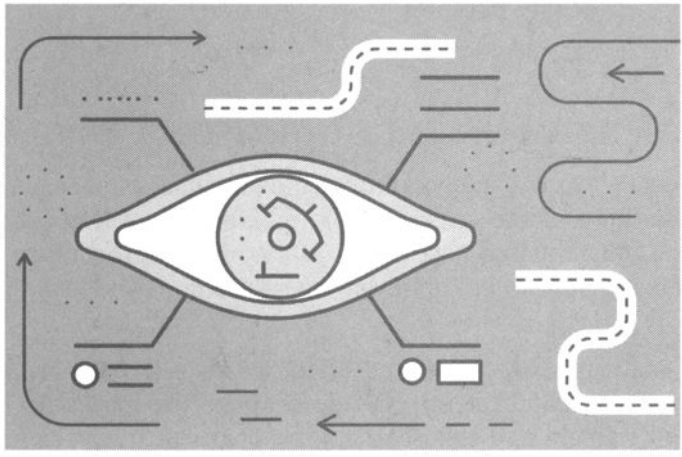


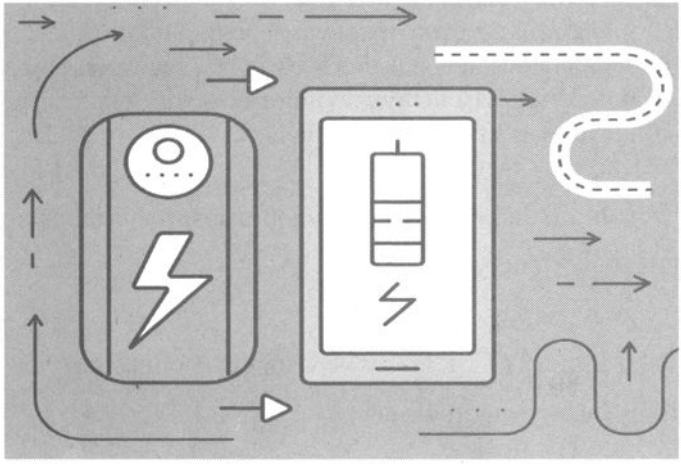
Рис. 13.2. Эффект прокрутки

Конечно, лучше все увидеть в действии, поэтому загрузить рабочий пример вы можете по адресу: **http://www.nit.com.ru**



Глава 14.

Введение в AJAX



14.1. Различные библиотеки JavaScript

Библиотека - это набор функций, выполняющих определенные действия. Например, вы можете создать свою библиотеку для обработки строк. В JavaScript большинство известных библиотек, с которым вам придется работать, так или иначе связаны с организацией пользовательского интерфейса.

Почему так и почему "придется"? Представьте, что вы написали довольно сложный сценарий, который превосходно отображается в IE, Chrome, Firefox, но в Opera или каком-то другом браузере имеются некоторые проблемы, и созданный интерфейс пользователя отображается не так, как задумано вами.

Ранее было отмечено, что разные веб-браузеры могут отображать HTML-код и интерпретировать JavaScript-сценарии по-разному. Протестировать все возможные браузеры одному разработчику, сами понимаете, нереально.

А теперь представьте, что вы используете библиотеку для организации интерфейса пользователя. Если в ней будет найдена ошибка (а ее найдут быстрее, поскольку этой библиотекой уже пользуются тысячи других программистов, и она используется в тысячах проектов, а не только в одном вашем), ее быстро исправят. Все, что вам нужно будет сделать для исправления этой ошибки в вашем проекте, - загрузить новую версию библиотеки. Вы можете даже не вдаваться в подробности и не вникать, что именно было исправлено, - просто загрузите новую версию библиотеки. Более того, для организации сложного интерфейса пользователя потребуются только основы знания JavaScript, вам не нужно понимать, что и как делает библиотека, а нужно лишь использовать предоставляемые ей методы. Все остальное библиотека выполнит за вас.

Существует много библиотек, но особенно хочется выделить следующие:

- jQuery (http://jquery.com).
- Prototype (www.prototypejs.org).
- Yahoo! UI Library (YUI, http://developer.yahoom.com/yui/).
- MooTools (http://mootools.net).
- Dojo (http://dojotoolkit.org).
- ExtJs (http://www.extjs.com).

Наиболее часто используемые библиотеки из этого списка - jQuery, MooTools и YUI. В следующей части этой книги будет рассмотрена библиотека jQuery, обеспечивающая кроссбраузерную поддержку (работает с IE, Chrome, Firefox, Opera и Safari). Библиотеку может использовать даже начинающий программист, освоивший только основы JavaScript. Кроме простоты использования популярности jQuery способствуют небольшой размер (библиотека очень компактна и не засоряет глобальное пространство имен ненужными идентификаторами, а ваш проект - тысячами файлов) и огромное количество всевозможных плагинов для этой библиотеки.

14.2. Введение в AJAX или перезагрузка страницы на лету

Представим себе обычную HTML-форму, например форму ввода имени пользователя и пароля. Форма с помощью параметра action ссылается на какой-то сценарий (он может быть написан на PHP, Perl, Python или любом другом языке программирования, но далее для большей однозначности мы будем предполагать, что сценарий написан на PHP). Как только пользователь нажимает кнопку Submit (название может быть другим), форма отправляет данные, введенные пользователем, PHP-сценарию, который их обрабатывает. При этом происходит перезагрузка: страница в браузере меняется - сначала вы видели страницу HTML-формы, затем - страницу PHP-сценария.

Некоторые современные сайты умеют передавать данные, введенные пользователем в форму, на сервер без перезагрузки страницы. Вы нажимаете Submit, данные отправляются на сервер, потом ваша форма входа превращается в набор служебных ссылок (вроде "Добавить статью", "Выход из системы" и т.д. - набор ссылок зависит от концепции сайта), а основная часть страницы превращается в ваш кабинет, где вы можете управлять материалами, которые добавили на сайт.

Как реализуется подобное поведение? При нажатии кнопки Submit вызывается JavaScript, который передает полученную информацию PHP-сценарию и получает от него ответ, потом (если пользователь прошел аутентификацию) этот же JavaScript перестраивает вашу текущую страницу. Получается, что страница осталась та же, но ее HTML-код изменился без перезагрузки.

Написать такой сценарий под силу программисту квалификации выше среднего. При особом желании и знании JavaScript вам тоже под силу реализовать подобную систему. Но делать этого не придется. К счастью, программисты с более высокой квалификацией создали технологию AJAX (Asynchronous Javascript and XML), позволяющую изменять код страницы без ее перезагрузки. Если вы знакомы с JavaScript, то можете возразить - мол, это умел и обычный JavaScript. Да, JavaScript может изменять код страницы без ее перезагрузки, но суть технологии AJAX заключается в поддержке баз данных. Новые данные, которые будут загружены в страницу, получаются из базы данных. А в случае с обычным JavaScript они являются частью скрипта, что немного не то.

14.3. Создание AJAX-приложения

Наше AJAX-приложение будет состоять из трех файлов:

- index.php - основной файл;
- ajax.js - содержит все необходимые сценарии на языке JavaScript;
- ajax.php - это и есть сценарий, к которому будет обращаться JavaScript-код.

Основной файл (index.php) не будет содержать PHP-код, а лишь HTML-форму, которая будет ссылаться на ajax.js. В нашем простом случае этого вполне достаточно. Но если вам нужно будет внедрить PHP-код, можете это сделать с помощью тегов <?php ?>. Все равно вы будете расширять функциональность вашего сценария, вот тогда PHP вам и пригодится.

Код из файла ajax.js выполняет обращение к файлу ajax.php, который производит обработку данных, полученных от ajax.js. Это может быть все что угодно - проверка имени пользователя и пароля, выборка из базы данных и т.д. В результате ajax.php сгенерирует XML-код, который будет передан обратно сценарию ajax.js. На основании полученного XML-кода ajax.js обновит HTML-код страницы без ее перезагрузки.

Вроде бы все понятно, осталось реализовать идею на практике. Начнем с файла index.php, который представлен в листинге 14.1.

Листинг 14.1. Файл index.php

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <title>Пример AJAX</title>
      <script type="text/javascript" src="ajax.js"></script>
</head>
<body onload='ajax()'>
      Введите свое имя:
      <input type="text" id="Name" />
      <div id="OurMessage" />
</body>
</html>
```

При загрузке страницы автоматически вызывается метод ajax() из файла ajax.js, который задан в теге <script>. Поле, в которое пользователь будет вводить свое имя, называется Name. Если вы надумали изменить его, то запомните, что вы указали в форме, - позже нужно будет это же имя указать в ajax.js. Тег с именем OurMessage будет принимать сообщение от сервера, то есть от сценария ajax.php.

У нас нет кнопки Submit, поскольку введенная информация будет каждую секунду отправляться на сервер, так что эта кнопка нам не нужна. В реальных проектах она может вам понадобиться - вряд ли пользователь сможет заполнить форму за 1 секунду. Тогда сделайте так, чтобы кнопка Submit запускала метод ajax() - не нужно запускать его при загрузке страницы:

```
<input type=BUTTON value="Submit" name="mySubmit" onClick="ajax()">
```

Вариант сценария index.php с кнопкой Submit представлен в листинге 14.2.

Листинг 14.2. Файл index.php, вариант 2

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>AJAX пример 2</title>
<script type="text/javascript" src="ajax.js"></script>
</head>
<body>
Введите ваше имя:
<input type="text" id="Name" />
```

```
<input type=BUTTON value="Submit" name="mySubmit"
onClick="ajax()">
<div id="OurMessage" />
</body>
</html>
```

Далее нам нужно написать файл ajax.js. Его задача - получить имя пользователя (значение текстового поля с именем Name) и передать его сценарию ajax.php. Код ajax.js приведен в листинге 14.3. Жирным выделены значения, на которые вам следует обратить внимание при модификации сценария.

Листинг 14.3. Файл ajax.js

```
// Объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// Получаем объект XMLHttpRequest
function createXmlHttpRequestObject()
{

var xmlHttp;
// Если используется Internet Explorer
if(window.ActiveXObject)
{
try
{
xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}
catch (e)
{
xmlHttp = false;
}
}
// Если используется другой браузер
else
{
try
{
xmlHttp = new XMLHttpRequest();
}
catch (e)
{
xmlHttp = false;
}
}
// Если не получилось создать объект XMLHttpRequest
```

```
if (!xmlHttp)
alert("Error creating the XMLHttpRequest object.");
else
return xmlHttp;
}
// Делаем HTTP-запрос
function ajax()
{

if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
{
// получаем имя пользователя, в форме полю ввода присвоено имя
Name
name = encodeURIComponent(document.getElementById("Name").
value);
// передаем введенное имя сценарию ajax.php
xmlHttp.open("GET", "ajax.php?name=" + name, true);

xmlHttp.onreadystatechange = processServerResponse;
// делаем запрос к серверу
xmlHttp.send(null);
}
else
// Если соединение занято, делаем повтор через 1 секунду
setTimeout('ajax()', 1000);
}
// Эта функция выполняется автоматически при получении ответа
от сервера
function processServerResponse()
{
if (xmlHttp.readyState == 4)
{
// статус 200 - транзакция прошла успешно
if (xmlHttp.status == 200)
{
// извлекаем XML, который мы получили от сервера
xmlResponse = xmlHttp.responseXML;
xmlDocumentElement = xmlResponse.documentElement;
helloMessage = xmlDocumentElement.firstChild.data;
// обновляем страницу: выводим полученный от ajax.php
результат
// в div с именем OurMessage
document.getElementById("OurMessage").innerHTML =
'<i>' + helloMessage + '</i>';
// перезапуск через 1 секунду
setTimeout('ajax()', 1000);
```

```
}
// если статус <> 200, значит, произошла ошибка
else
{
alert("Ошибка доступа к серверу: " +
xmlHttp.statusText);
}
}
}
```

Как видите, полученное значение текстового поля передается сценарию ajax.php для дальнейшей обработки. Чтобы вы не запутались с именами переменных, рекомендую называть переменные одинаково во всех трех файлах. Если в форме переменная называется name, то в сценарии JavaScript для внутреннего имени тоже используйте переменную с именем name, сценарию ajax.php тоже нужно передать GET-переменную с именем name - как будто бы она пришла непосредственно из формы ввода.

Наш сценарий ajax.php никакой особенной обработки производить не будет, а просто выведет имя пользователя и пожелает ему приятного дня (лист. 14.4.)

Листинг 14.4. Файл ajax.php

```
<?php
// XML output
header('Content-Type: text/xml');
// XML header
echo '<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>';
// print the <response> element
echo '<response>';
// получаем имя пользователя
$name = $_GET['name'];

echo htmlentities($name) . ', хорошего Вам дня!';
// конец <response>
echo '</response>';
?>
```

Результат работы сценария представлен на рис. 14.1. Понимаю, что сценарии довольно объемны (особенно ajax.js), поэтому для улучшения восприятия рекомендуется все "пощупать" лично.

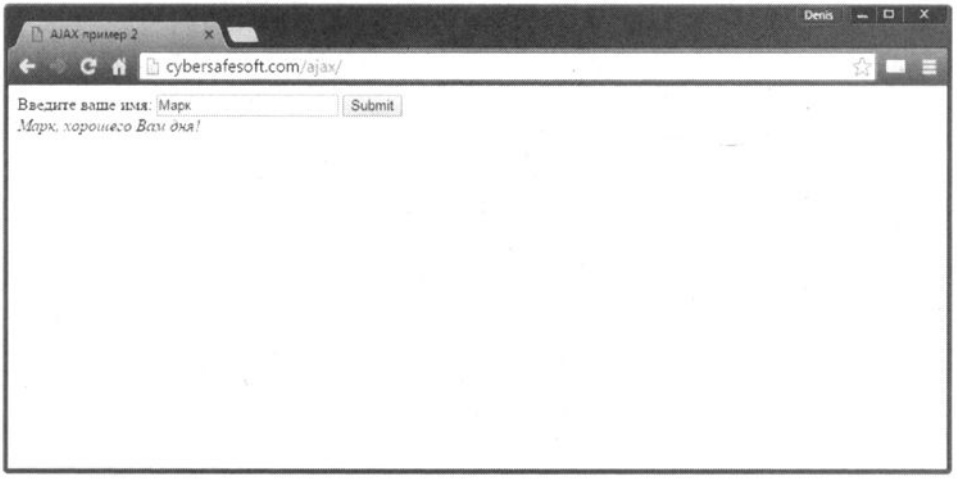
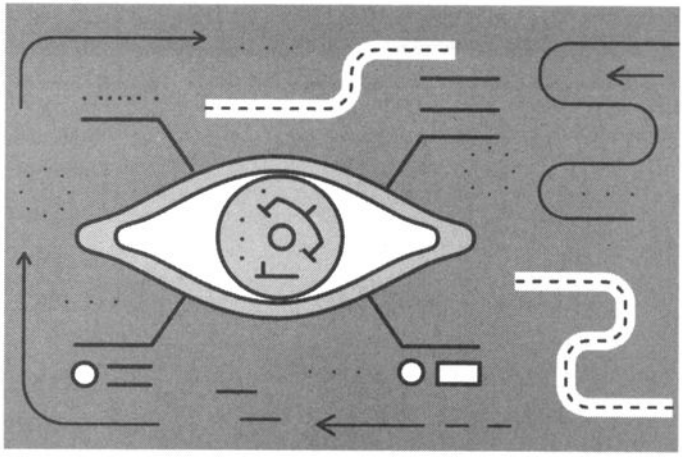


Рис. 14.1. Результат работы AJAX-приложения

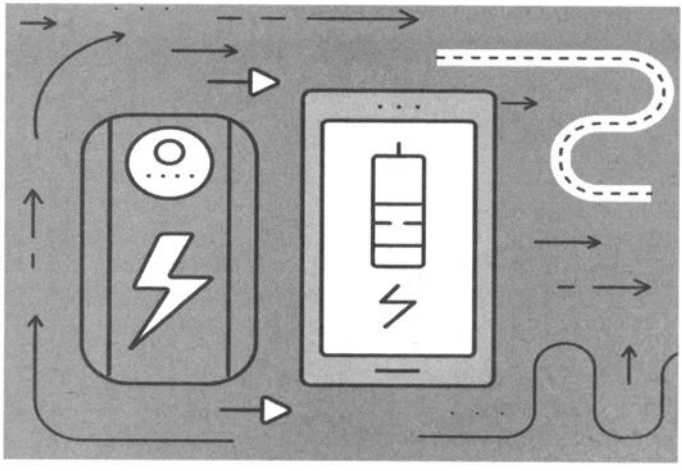
Внимание!

Чтобы у вас не было проблем с кодировками, всегда используйте кодировку UTF-8. Ее поддерживают все современные браузеры и текстовые редакторы. Файлы с кодом сохраняйте в кодировке UTF-8, данные в базе данных храните также в этой кодировке. Все текстовые редакторы, предназначенные для редактирования кода, даже простейший Notepad2, поддерживают кодировку UTF.



Глава 15.

Добавляем jQuery UI на сайт



Ранее было показано несколько примеров, построенных на основании библиотеки jQuery. Как правило, это были различные плагины для jQuery. В этой главе мы рассмотрим собственные возможности этой библиотеки по созданию интерфейса пользователя.

15.1. Загрузка jQuery UI

Скачать библиотеку UI jQeery можно совершенно свободно (без оплаты и регистрации) с сайта http://jqueryui.com/download/. На данный момент последней является версия 1.12 для jQuery 1.7. Именно эту версию и рекомендуется использовать.

При загрузке библиотеки можно выбрать тему оформления (рис. 15.1). Хотя вы можете загрузить другую тему в любое удобное для вас время. На иллюстрациях в этой книге будет использоваться тема оформления UI lightness.

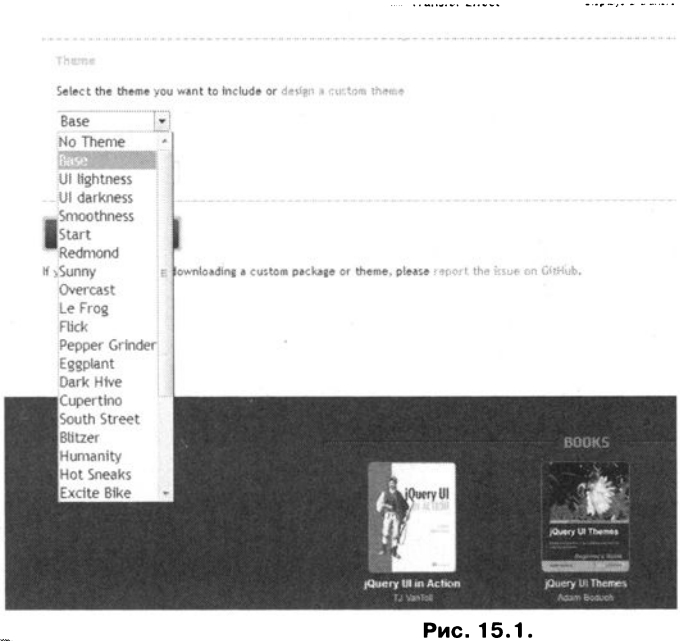


Рис. 15.1.

15.2. Выбор даты

Далее мы рассмотрим ряд виджетов и покажем, какие из них вы можете использовать при разработке сайтов разной сложности.

Виджет - это уже готовый элемент интерфейса пользователя. Библиотека UI jQuery содержит целый набор различных элементов пользовательского интерфейса. Например, в этом разделе будет рассмотрен элемент выбора даты. А это означает, что вам никогда не придется разрабатывать его самостоятельно - вы можете использовать уже готовое решение.

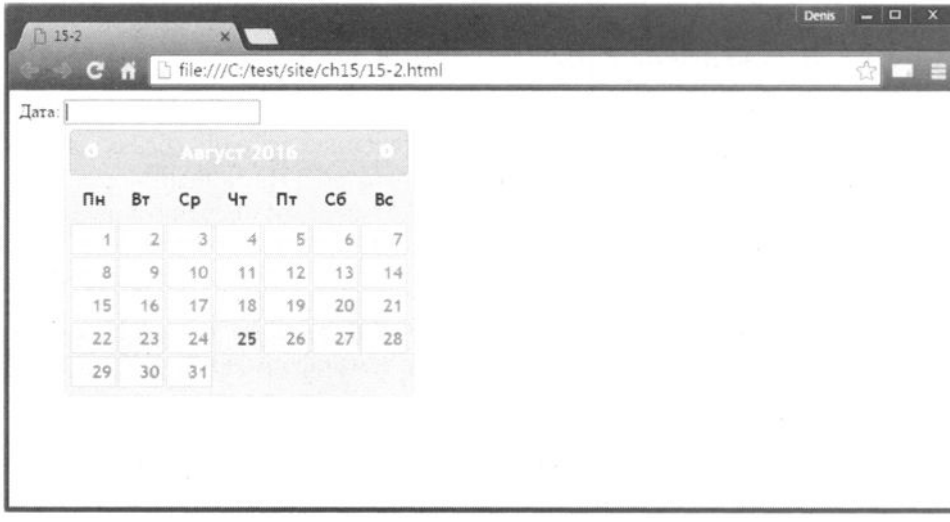


Рис. 15.2. Виджет DataPicker

Посмотрите на рис. 15.2. На нем изображен виджет DataPicker из библиотеки UI jQuery. Сейчас и вы научитесь создавать такой же. Представим, что у нас есть поле ввода даты:

```
<p>Date: <input id="date" type="text" /></p>
```

Ничего особенного, обычное поле с id="date". Вызвать виджет DataPicker можно с помощью метода datapicker():

```
$("#date").datepicker();
```

Вот только чтобы все заработало как нужно, необходимо подключить библиотеку UI jQuery. Как уже отмечалось ранее, при загрузке библиотеки можно выбрать тему оформления. Доступно много подобных тем, к сожалению, нельзя скачать их все сразу, приходится довольствоваться несколькими вариантами, а потом извлечь из них необходимые вам CSS-файлы.

Подключить необходимую тему оформления можно как обычный CSS-файл, например:

```
<link   type="text/css"   href="css/название-темы/jquery-ui.
custom.css" rel="stylesheet" />
```

Подключение всего необходимого (не только для этого, но и для всех последующих примеров в этой главе) выглядит так:

```
<link   type="text/css"   href="css/ui-lightness/jquery-ui.
custom.css" rel="stylesheet" />
<script src="js/jquery.js" type="text/javascript"></script>
<script     src="js/jquery-ui.custom.min.js"     type="text/
javascript"></script>
<script   src="js/jquery-ru.js"   type="text/javascript"></
script>
```

Код раскрытия календаря следующий:

```
<script type="text/javascript">
$(function(){
            $.datepicker.setDefaults($.extend($.datepicker.
regional['ru']));
  $("#date").datepicker();
});
</script>
```

Обратите внимание, что при открытии календаря мы указываем локализацию - 'ru'. Однако все равно календарь был на английском. Чтобы его русифицировать, нужно создать файл jquery-ru.js (мы его подключаем к нашей странице, см. выше), приведенный в листинге 15.1.

Листинг 15.1. Файл локализации jquery-ru.js

```
jQuery(function($){
    $.datepicker.regional['ru'] = {
```

```
    monthNames: ['Январь', 'Февраль', 'Март', 'Апрель',
    'Май', 'Июнь', 'Июль', 'Август', 'Сентябрь',
    'Октябрь', 'Ноябрь', 'Декабрь'],
    dayNamesMin: ['Вс','Пн','Вт','Ср','Чт','Пт','Сб'],
    firstDay: 1,
    };
    $.datepicker.setDefaults($.datepicker.regional['ru']);
});
```

Данный файл вы не найдете в комплекте jQuery UI.

Полный исходный код страницы примера приведен в листинге 15.2

Листинг 15.2. Полный код страницы выбора даты

```
<html>
<head>
<title>15-2</title>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<link type="text/css" href="css/ui-lightness/jquery-ui.custom.
css" rel="stylesheet" />
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.custom.min.js" type="text/
javascript"></script>
<script src="js/jquery-ru.js" type="text/javascript"></script>
<script type="text/javascript">
$(function(){
 $.datepicker.setDefaults($.extend($.datepicker.
regional['ru']));
  $("#date").datepicker();
});
</script>
</head>
<body>
<p>Дата: <input id="date" type="text" /></p>

</body>
</html>
```

15.3. Диалоговое окно

С помощью виджета Dialog можно организовать привлекательное диалоговое окно, которое затем перемещать по странице и при необходимости

закрыть. Также пользователь может изменять размер такого диалогового окна. Очень полезная штука при выводе различных уведомлений или рекламных блоков.

Организовать такое окно очень просто. Для начала нужно задать текст, который будет использоваться для помещения в диалоговое окно:

```
<div id="dialog" title="Привет!">
  <p>Только сегодня! Успей купить гироскутер всего за 300$ и
получи в подарок шлем!</p>
```

В данном случае в качестве текста будет использован текст из с идентификатором "dialog", а атрибут title будет задействован в качестве заголовка окна.

Далее нужно вызвать метод dialog():

```
$("#dialog").dialog();
```

Собственно, вот и все. Полный исходный код приведен в листинге 15.3, а результат работы метода dialog() - на рис. 15.3.

Листинг 15.3. Использование метода dialog()

```
<html>
<head>
<title>Диалог</title>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<link type="text/css" href="css/ui-lightness/jquery-ui.custom.
css" rel="stylesheet" />
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.custom.min.js" type="text/
javascript"></script>
<script type="text/javascript">
$(function(){
  $("#dialog").dialog();
});
</script>
</head>
<body>
<div id="dialog" title="Привет!">
  <p>Только сегодня! Успей купить гироскутер всего за 300$ и
получи в подарок шлем!</p>
</div>
```

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

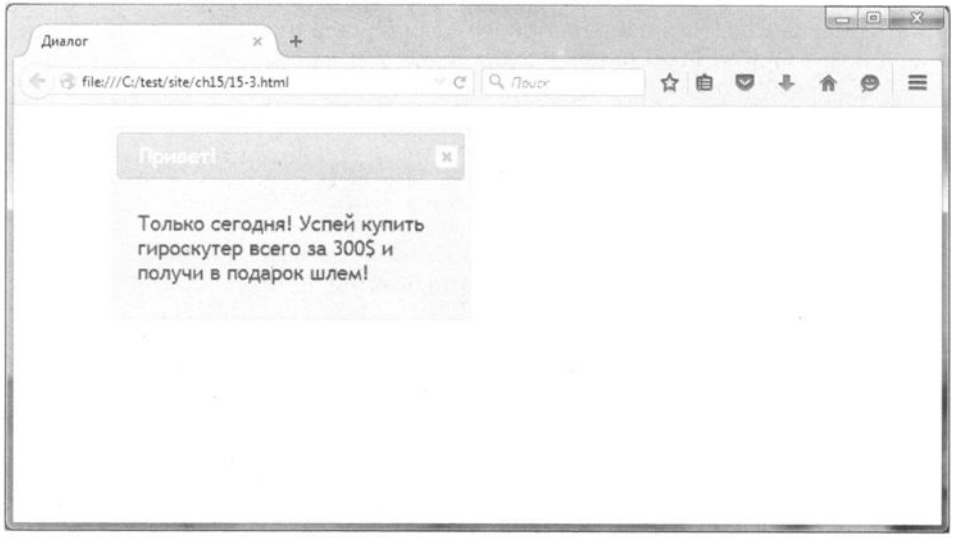


Рис. 15.3. Созданное диалоговое окно

15.4. Раскрывающиеся секции

Виджет Accordion позволяет организовать раскрывающиеся секции HTML-кода. Чтобы вам было понятно, что мы создаем, взгляните на рис. 15.4. Думаю, вы не раз видели подобные виджеты на других сайтах. Оказывается, организовать такой виджет очень просто.

Первым делом нужно определить , содержащий подходящую для Accordion структуру. А именно: заголовки секций должны быть оформлены как <h3>, а после заголовка должен следовать , содержащий текст раздела (обязательно должен быть закрыт!). Внутри можно использовать любые HTML-операторы: создавать списки, вставлять картинки, формировать таблицы и т.д.

Вот пример нашего с необходимой информацией:

```
<div id="accordion">
  <h3><a href="#">Описание</a></h3>
  <div>
```

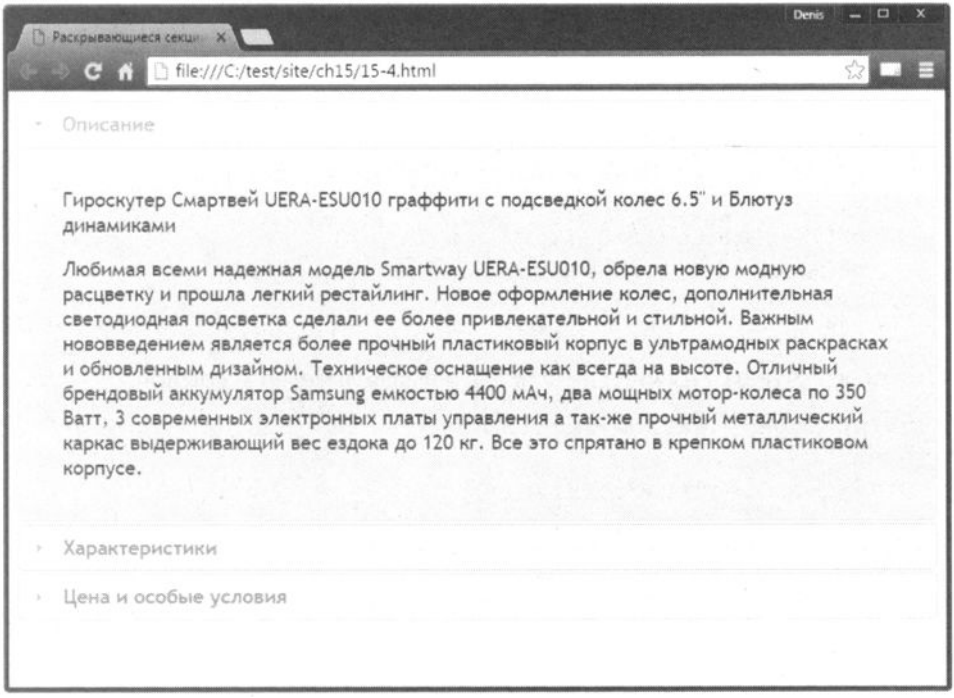


Рис. 15.4. Виджет Accordion в действии

```
    <p>Гироскутер Смартвей UERA-ESU010 граффити  с подсветкой
колес 6.5" и блютуз-динамиками</p>

     <p>Любимая всеми надежная модель Smartway UERA-ESU010
обрела новую модную расцветку и прошла легкий рестайлинг.
Новое оформление колес, дополнительная светодиодная подсветка
сделали ее более привлекательной и стильной. Важным
нововведением является более прочный пластиковый корпус в
ультрамодных раскрасках и с обновленным дизайном. Техническое
оснащение как всегда на высоте. Отличный брендовый аккумулятор
Samsung емкостью 4400 мАч, два мощных  мотор-колеса по 350
Ватт, 3 современных электронных платы управления а также
прочный металлический каркас, выдерживающий вес ездока до
120 кг. Все это спрятано в крепком пластиковом корпусе.</p>
  </div>
  <h3><a href="#">Характеристики</a></h3>
  <div>
    <p>Максимальная нагрузка: 120 кг
    <p>Мощность: 700W (350W * 2)
    <p>Время зарядки: 2-2,5 ч
```

```
    <p>Пробег на одной зарядке: 20-25 km
    <p>Емкость батареи: 4400mAh 158w (батарея Samsung)
    <p>Размер колес: 6,5 дюймов  (18 см)
    <p>Влагозащита: IP54 </p>
  </div>
  <h3><a href="#">Цена и особые условия</a></h3>
  <div>
    <p>Обычная цена: $375
    <p>Акционная (только до 31.08.2016): $299
  </div>
```

Потом, все, что вам нужно сделать, - это вызвать метод accordion:

```
<script type="text/javascript">
$(function(){
  $("#accordion").accordion();
});
</script>
```

Собственно, вот и все. Полный код страницы приведен в листинге 15.4.

Листинг 15.4. Использование виджета Accordion

```
<html>
<head>
<title>Раскрывающиеся секции</title>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<link type="text/css" href="css/ui-lightness/jquery-ui.custom.
css" rel="stylesheet" />
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.custom.min.js" type="text/
javascript"></script>
<script type="text/javascript">
$(function(){
  $("#accordion").accordion();
});
</script>
</head>
<body>
<div id="accordion">
  <h3><a href="#">Описание</a></h3>
  <div>
```

```
    <p>Гироскутер Смартвей UERA-ESU010 граффити  с подсветкой
колес 6.5" и блютуз-динамиками</p>

    <p>Любимая всеми надежная модель Smartway UERA-ESU010
обрела новую модную расцветку и прошла легкий рестайлинг.
Новое оформление колес, дополнительная светодиодная
подсветка  сделали ее более привлекательной и стильной. Важным
нововведением является более прочный пластиковый корпус в
ультрамодных раскрасках и с обновленным дизайном. Техническое
оснащение как всегда на высоте. Отличный брендовый аккумулятор
Samsung емкостью 4400 мАч, два мощных  мотор-колеса по 350
Ватт, 3 современных электронных платы управления а также
прочный металлический каркас, выдерживающий вес ездока до 120
кг. Все это спрятано в крепком пластиковом корпусе.</p>
  </div>
  <h3><a href="#">Характеристики</a></h3>
  <div>
    <p>Максимальная нагрузка: 120 кг
    <p>Мощность: 700W (350W * 2)
    <p>Время зарядки: 2-2,5 ч
    <p>Пробег на одной зарядке: 20-25km
    <p>Емкость батареи: 4400mAh 158w ( батарея Samsung)
    <p>Размер колес: 6,5 дюймов  (18 см)
    <p>Влагозащита: IP54 </p>
  </div>
  <h3><a href="#">Цена и особые условия</a></h3>
  <div>
    <p>Обычная цена: $375
    <p>Акционная (только до 31.08.2016): $299
  </div>

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

15.5. Индикатор процесса

Наверняка вы видели индикатор процесса, показывающий ход выполнения какого-то процесса, например загрузки файла.

Индикатор процесса создается методом progressbar(). При изменении значения индикатора (позже я покажу, как это сделать) генерируется событие progressbarchange. Когда значение индикатора достигнет 100, будет сгенерировано событие progressbarcomplete.

В листинге 15.5 приведен полный код страницы, демонстрирующей управление индикатором процесса.

Листинг 15.5. Демонстрация управления индикатором процесса

```
<html>
<head>
<title>29-4</title>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<link type="text/css" href="css/ui-lightness/jquery-ui.custom.
css" rel="stylesheet" />
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.custom.min.js" type="text/
javascript"></script>
<script type="text/javascript">
$(function(){
  $("#bar").progressbar({
    value: 0,
    change: function(event, ui) {
     //window.alert('Event ' + event.type);
    },
    complete: function(event, ui) {
     window.alert('Event ' + event.type);
    }
  });
  $("button").click(function(){
   var currentVal = $("#bar").progressbar('option', 'value');

   if(currentVal<100) {
     currentVal = currentVal + 10;
     $("#percent").text(currentVal + "%");
      $("#bar").progressbar('option', 'value', currentVal);

   }
  });
});
</script>
</head>
<body>
<div id="bar"></div>
<button>Добавить 10</button>
<div id="percent"></div>
</body>
</html>
```

У нас есть кнопка, увеличивающая значение индикатора процесса на 10. В с идентификатором "result". При ее нажатии мы первым делом получаем текущее значение индикатора в переменную currentVal, затем если это значение еще не достигло 100, мы увеличиваем его на 10 (в реальном мире придется увеличивать на столько, сколько работы по-настоящему выполнено), выводим в область result и устанавливаем новое значение методом progressbar().

Как только значение индикатора достигнет 100, будет сгенерировано событие progressbarcomplete и будет запущен его обработчик - вы увидите диалоговое окно с типом события. А вот если вы хотите (из соображений отладки, конечно) видеть подобное диалоговое окно при каждом изменении значения индикатора, раскомментируйте выделенный жирным оператор. Пример работающего индикатора процесса приведен на рис. 15.5.

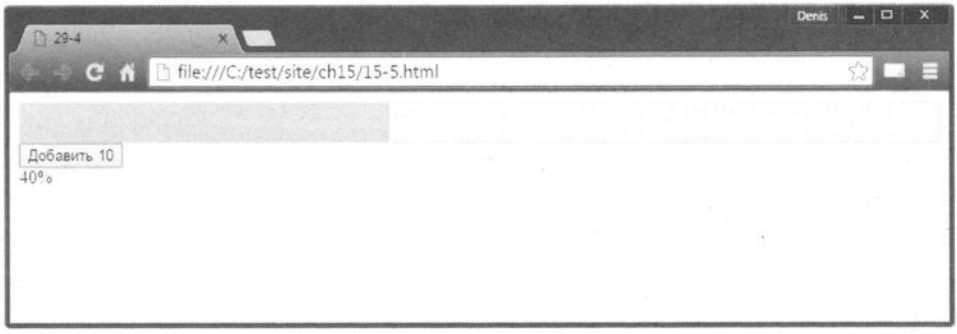


Рис. 15.5. Индикатор процесса

Вообще обработчик события progressbarchange можно использовать для чего-то полезного, например, если индикатор загрузки отображает процесс установки вашего программного продукта, то можете менять слайды, демонстрируя пользователю особенности вашего продукта. Все мы устанавливали хоть раз в жизни Windows, поэтому, думаю, вы понимаете, о чем я говорю. Конечно, на JavaScript и PHP операционную систему не напишешь, но зато можно написать систему управления контентом и сделать для нее красочный инсталлятор, а в этом как раз вам поможет UI jQuery.

15.6. Вкладки

Иногда удобнее представить информацию во вкладках, например, когда вы разрабатываете интерфейс сценария, изменяющий параметры вашего программного продукта. Вместо того чтобы создавать несколько страниц с настройками, вы можете использовать одну страницу, но несколько вкладок (см. рис. 15.6).

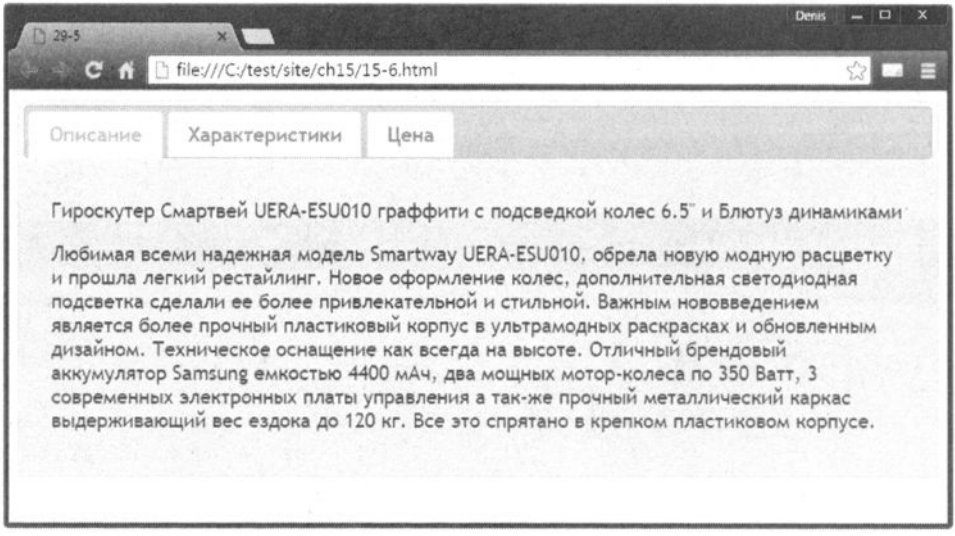


Рис. 15.6. Вкладки, реализованные с помощью UI jQuery

Организовать вкладки очень просто. Для этого нужно создать список, состоящий из заголовков вкладок, а затем в отдельных блоках (id блока должен соответствовать id вкладки) описать содержимое каждой вкладки:

```
<div id="tabs">
  <ul>
    <li><a href="#tabs-1">Описание</a></li>
    <li><a href="#tabs-2">Характеристики</a></li>
    <li><a href="#tabs-3">Цена</a></li>
  </ul>
  <div id="tabs-1">
    <p>Содержимое 1-й вкладки </p>
  </div>
```

```
  <div id="tabs-2">
    <p>Содержимое 2-й вкладки</p>
  </div>
  <div id="tabs-3">
    <p>Содержимое 3-й вкладки</p>
  </div>
</div>
```

После того как вкладки определены, нужно вызвать метод tabs() для родительского блока:

```
  $("#tabs").tabs();
```

Полный код страницы с вкладками представлен в листинге 15.6.

Листинг 15.6. Вкладки с помощью UI jQuery

```
<html>
<head>
<title>29-5</title>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<link type="text/css" href="css/ui-lightness/jquery-ui.custom.
css" rel="stylesheet" />
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.custom.min.js" type="text/
javascript"></script>
<script type="text/javascript">
$(function(){
  $("#tabs").tabs();
});
</script>
</head>
<body>
<div id="tabs">
  <ul>
    <li><a href="#tabs-1">Описание</a></li>
    <li><a href="#tabs-2">Характеристики</a></li>
    <li><a href="#tabs-3">Цена</a></li>
  </ul>
  <div id="tabs-1">
    <p>Содержимое 1-й вкладки.</p>
  </div>
  <div id="tabs-2">
    <p>Содержимое 2-й вкладки</p>
```

```
  </div>
  <div id="tabs-3">
    <p>Содержимое 3-й вкладки</p>
  </div>
</div>

</body>
</html>
```

15.7. Кнопки

Описание библиотеки UI jQuery было бы не полным, если бы мы не рассмотрели, какие она умеет формировать кнопки. Казалось, нужно было с кнопок и начать, поскольку они всегда считаются самыми примитивными виджетами, однако хотелось сразу начать с чего-то более интересного и полезного. Тем более, кнопки UI jQuery мне кажутся немного вычурными (рис. 15.7). С одной стороны, симпатично, с другой они впишутся далеко не в каждый дизайн, поэтому придется экспериментировать с CSS-кнопками или же с другой темой оформления для всего UI jQuery.

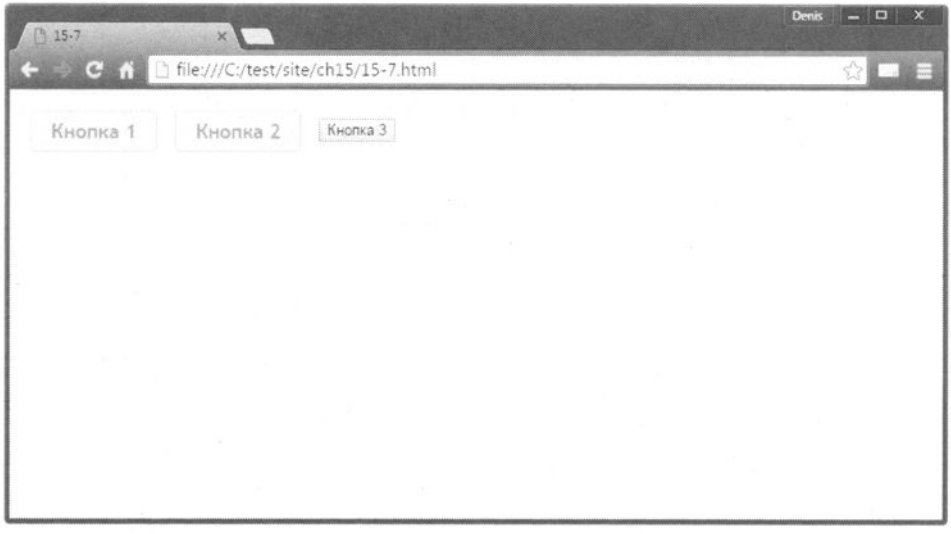


Рис. 15.7. Виджет Button

Рассмотрим код страницы, изображенной на рис. 15.7 (лист. 15.7).

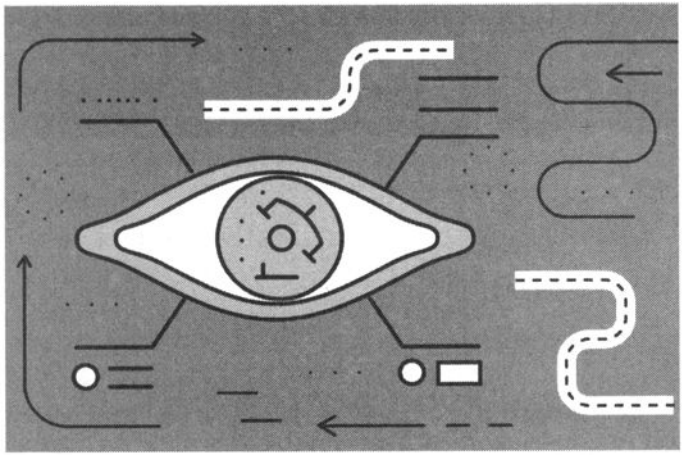
Листинг 15.7. Код страницы с кнопками

```
<html>
<head>
<title>15-7</title>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<link type="text/css" href="css/ui-lightness/jquery-ui.custom.
css" rel="stylesheet" />
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.custom.min.js" type="text/
javascript"></script>

<style type="text/css">
button { margin:10px; }
</style>

<script type="text/javascript">
$(function(){
  $("#button1").button().click(function(e){
    alert("Button 1 pressed");

  });

   $("#button2").button().click(function(e){
    alert("Button 2 pressed");

  });
});
</script>
</head>
<body>
<div>
  <button id="button1">Кнопка 1</button>
  <button id="button2">Кнопка 2</button>
  <button id="button3">Кнопка 3</button>
</body>
</html>
```

Нужно отметить, что мало просто подключить библиотеку UI jQuery, чтобы все кнопки превратились в кнопки, показанные на рис. 15.7 (имеются в виду первые две кнопки). Как видно из кода, представленного в лист. 15.7, для первых двух кнопок вызывался метод button(). Именно он и выполня-

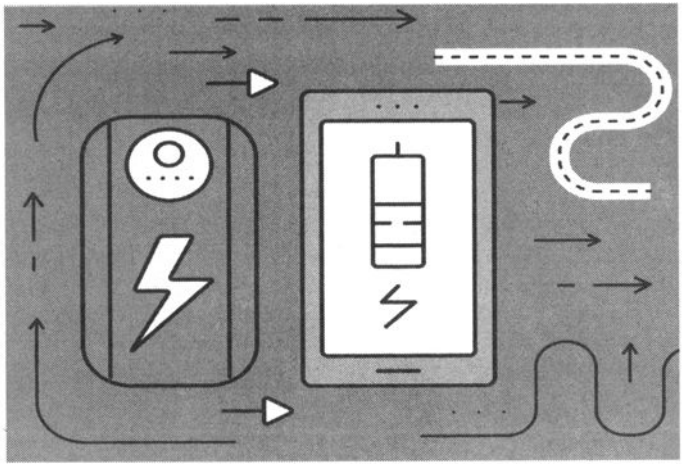
ет превращение обычной кнопки в UI-кнопку. Для третьей кнопки мы этот метод не вызывали, и она выглядит как обычно.

На этом всё. Если вы заинтересовались, получить информацию о дополнительных виджетах, а также испытать их в действии вы всегда можете по адресу: http://jqueryui.com/demos/.



Глава 16.

Наворачиваем изображения



16.1. Изменение изображения по событию мыши

Используя события OnMouseOver и OnMouseOut, а также OnMouseDown, можно изменять картинки на странице. Один из самых наиболее распространенных сценариев - изменить картинку при наведении на изображение указателя мыши (событие OnMouseOver), а затем восстановить первоначальное изображение при событии OnMouseOut. Рассмотрим пример (лист. 16.1).

Листинг 16.1. Изменение изображения при наведении мыши

```
<a href="#"
       OnMouseOver="document.getElementById('scooter').
src='yellow.png'"
       OnMouseOut="document.getElementById('scooter').
src='blue.png'">
        <img id="scooter" src="blue.png" border=0
align="left"></a>
```

Код очень прост: в элемент страницы с идентификатором scooter загружается изображение синего гироскутера blue.png. При наведении на него мыши изображение заменяется на изображение желтого скутера - yellow.png. Как только указатель мыши выйдет за пределы изображения, будет восстановлено исходное изображение.

Аналогично можно было заменить изображение по нажатию на него мышью. Для этого нужно использовать событие OnMouseDown - оно происходит, когда пользователь нажимает кнопку мыши. Восстановить исходное изображение можно по событию OnMouseUp (лист. 16.2).

Листинг 16.2. Изменение изображения при нажатии кнопки мыши

```
<a href="#"
```

Рис. 16.1. Замена изображения по событию мыши

```
       OnMouseDown="document.getElementById('scooter').
src='yellow.png'"
       OnMouseUp="document.getElementById('scooter').
src='blue.png'">
        <img id="scooter" src="blue.png" border=0
align="left"></a>
```

16.2. Слайдер в виде фотопленки

В главе 5 мы создали два слайдера - один был попроще и написан средствами обычного JavaScript, второй - с помощью библиотеки jQuery. В этой главе мы разработаем еще один довольно эффектный слайдер в виде фотопленки. На рис. 16.2 показано, что у нас должно получиться в итоге.

Первым делом начнем разработку таблицы стилей для нашего слайдера:

```
<style type="text/css">
...
    .scroller_row {
        position: relative;
        white-space: nowrap;
        width: 200%;
        padding: 22px 0 22px 0;
```

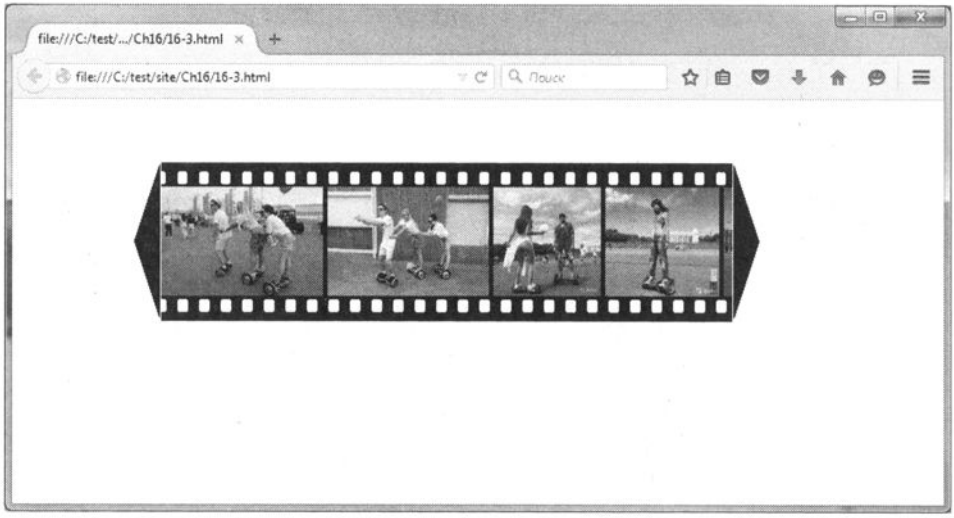


Рис. 16.2. Слайдер в виде фотопленки

```
        background: url('images/foto.gif') repeat-x;
    }
 ...
</style>
```

Основной момент в таблице стилей выделен жирным. Если у вас в слайдере много изображений, то вам нужно увеличить значение параметра width, иначе фотопленка (фоновое изображение foto.gif) не будет дублироваться.

Мы предполагаем, что высота изображений в слайдере будет равна 100 пикселям. Если вы захотите увеличить это изображение, тогда вам нужно предоставить другое фоновое изображение слайдера - чтобы оно могло поместить изображения увеличенного размера.

Полный код примера (CSS, JavaScript и HTML) вместе с комментариями приведен в листинге 16.3.

Листинг 16.3. Полный код фото-слайдера

```
<head>
<style type="text/css">
    .scroller_container {
        padding-left:100px;
        padding-top:50px;
    }
    .scroller_container div {
```

```
        float: left;
        vertical-align: baseline;
    }
    .scroller_window {
        width: 530px;
        overflow: hidden;
    }
    .scroller_row {
        position: relative;
        white-space: nowrap;
        width: 200%;
        padding: 22px 0 22px 0;
        background: url('images/foto.gif') repeat-x;
    }
    .scroller_row img {
        padding: 0;
        margin:0;
    }
    .scroller_row a {
        padding:0;
        margin:0;
    }
    .scroller_container > IMG{
        float: left;
        vertical-align: baseline;
        padding: 2px 1px 0 1px;
        cursor: pointer;
    }
    a img {
        border:0;
    }
</style>

<script type="text/javascript">
    var _scroller;
    _scroller = function () { // слайдер
        return{
            speed:10,           /*скорость в миллисекундах*/
            position:0,
            width:0,
            step:0,
            t:null,
            window:0,
            // Инициализация слайдера
            init: function () {
                var el;
```

```
            // Получить ширину внутренней части слайдера
            el = document.getElementById('scroller_rule');
            _scroller.width = el.clientWidth;
            // Получить ширину окна слайдера
            el = document.getElementById('scroller_
window');
            _scroller.window = el.clientWidth;

            // Установка обработчика колесика мыши
            addEvent(el, 'mousewheel', this.wheel);
            addEvent(el, 'DOMMouseScroll', this.wheel);

            // fix размеров для background (Opera, Chrome)
            el = document.getElementById('scroller_row');
            el.style.width = _scroller.width+'px';
        },

        // Обработчик колесика мыши
        wheel: function (e) {
            // Если уже идет прокрутка, то игнорировать событие
            if (_scroller.t != null) {
                _scroller.stop();
            }

            e = e ? e : window.event;
        var wheelElem = e.target ? e.target : e.srcElement;
        var wheelData = e.detail ? e.detail * -1 : e.wheelDelta / 40;

        // В движке WebKit возвращается значение в 100 раз больше
            if (Math.abs(wheelData) > 100) {
                wheelData = Math.round(wheelData / 100);
            }
            if (wheelData < 0) {
                _scroller.step = 5 * Math.abs(wheelData);
                _scroller.timer('right', 1);
            }
            else {
                _scroller.step = 5 * Math.abs(wheelData);
                _scroller.timer('left', 1);
            }
            if (window.event) {
                e.cancelBubble = true;
                e.returnValue = false;
                e.cancel = true;
            }
            if (e.stopPropagation && e.preventDefault) {
```

```
                e.stopPropagation();
                e.preventDefault();
            }
            return false;
        },

        // Функция скроллера
        scroll: function (dir, wheel) {
            var el = document.getElementById('scroller_row');
            var step;
            if (wheel == 0) {
                step = 2;
            }else{
             step = Math.round(Math.log(_scroller.step * 2) * 2.5);
                _scroller.step--;

                if (_scroller.step == 0) {
                    _scroller.stop();
                }
            }
            if (dir == 'left') {
                _scroller.position += step;
                if (_scroller.position > 0) {
                    _scroller.position = 0;
                    _scroller.stop();
                }
            }
            else {    // dir='rigth' (направление вправо)
                _scroller.position -= step;
                if (_scroller.position < (_scroller.window
- _scroller.width)) {
                    _scroller.position = _scroller.window
- _scroller.width;
                    _scroller.stop();
                }
            }
            el.style.left = _scroller.position + 'px';
        },

        // Таймер слайдера
        timer: function (dir, wheel) {
            _scroller.stop();
            _scroller.t = setInterval("_scroller.scroll('"
+ dir + "', " + wheel + ");", _scroller.speed);
        },
```

```
            // Остановка слайдера
            stop: function () {
                if (_scroller.t != null) {
                    clearInterval(_scroller.t);
                    _scroller.t = null;
                }
            }
        };
    }();

    window.onload=function(){setTimeout(_scroller.init,100);};
// задержка при запуске
</script>
</head>
<body>

<div class="scroller_container">
    <img src="images/left.gif" alt="" onmousemove="_scroller.
timer('left',0);" onmouseout="_scroller.stop();">
    <div class="scroller_window" id="scroller_window">
        <div class="scroller_row" id="scroller_row">
            <div id="scroller_rule">
                <a href="#1" onclick="return!1"><img src="s1.
jpg" height="100" alt=""></a>
                <a href="#2" onclick="return!1"><img src="s2.
jpg" height="100" alt=""></a>
                <a href="#3" onclick="return!1"><img src="s3.
jpg" height="100" alt=""></a>
                <a href="#4" onclick="return!1"><img src="s4.
jpg" height="100" alt=""></a>
                <a href="#5" onclick="return!1"><img src="s5.
jpg" height="100" alt=""></a>
                <a href="#6" onclick="return!1"><img src="s6.
jpg" height="100" alt=""></a>
                <a href="#7" onclick="return!1"><img src="s7.
jpg" height="100" alt=""></a>
                <a href="#8" onclick="return!1"><img src="s8.
jpg" height="100" alt=""></a>
                <a href="#9" onclick="return!1"><img src="s9.
jpg" height="100" alt=""></a>


            </div>
        </div>
    </div>
```

```
    <img src="images/right.gif" alt="" onmousemove="_scroller.
timer('right',0);" onmouseout="_scroller.stop();">
</div>
```

При щелчке на изображении ничего не происходит благодаря установке события OnClick:

```
onclick="return!1"
```

Нажатие кнопок со стрелками влево и вправо обеспечивает переключение фотографий в слайдере:

```
<img src="images/right.gif" alt="" onmousemove="_scroller.
timer('right',0);" onmouseout="_scroller.stop();">
```

К преимуществам нашего способа можно отнести то, что наш слайдер не требует сторонних библиотек.

16.3. Загрузка изображения в div

В этом примере будет показано, как загрузить изображение в DIV. Во время загрузки изображения будет отображаться надпись "Загружается", пока изображение не будет загружено. Полный код приведен в листинге 16.4, а результат его выполнения изображен на рис. 16.3.

Листинг 16.4. Код примера

```
<html>
<head>
<script>
function imgdiv(url)
    {
    var el=document.getElementById('img_div');
    var img=new Image();
    img.onload=function()
        {
        el.style.width=img.width+'px';
        el.style.height=(img.height+20)+'px';
```

```
        el.innerHTML='<img src='+url+' style="margin:0"
width="'+img.width+'" height="'+img.height+'" /><br> ('+img.
width+'x'+img.height+')';
        }
    el.innerHTML='Загружается...';
    img.src=url;
    }

</script>
</head>
<body>
<h3><a href='#' onclick='imgdiv("blue.png")'>Открыть DIV</a></
h3>

<div id="img_div" style='BORDER:#9fbddd 1px solid;'></div>
</body>
</html>
```

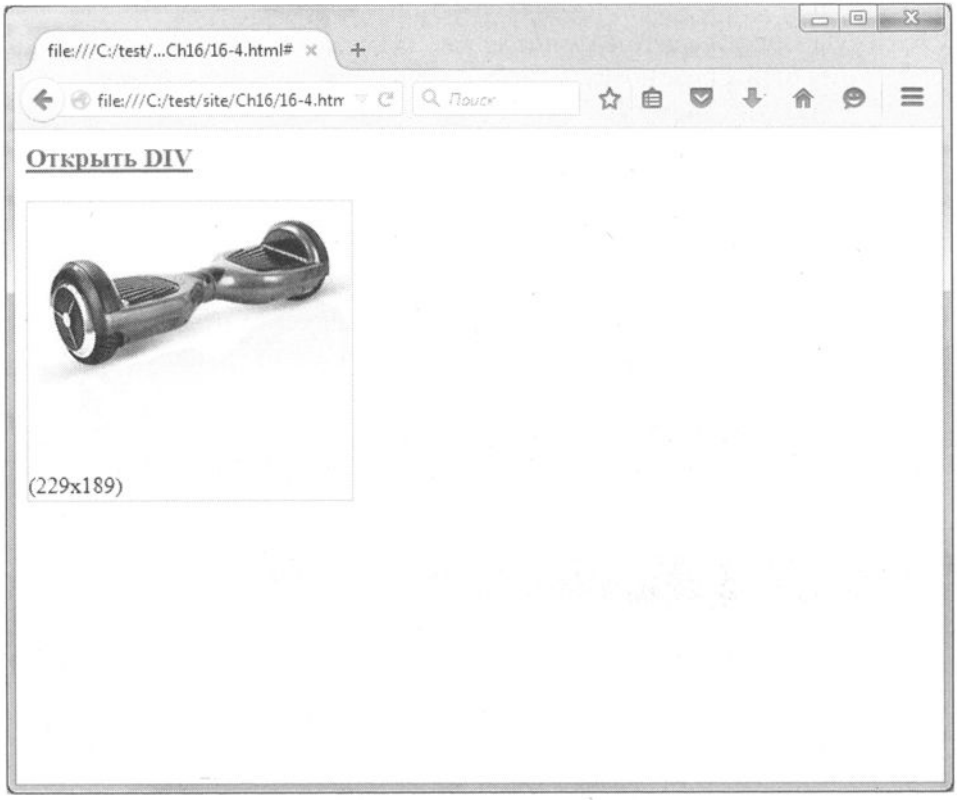


Рис. 16.3. Щелчок на надписи приводит к загрузке изображения в DIV

16.4. Карусель фотографий

После простого примера настало время опять для более сложного. На этот раз мы реализуем карусель фотографий. Фотографии, загруженные в карусель, будут прокручиваться непрерывно, пока пользователь не наведет указатель мыши на карусель.

В качестве основы мы будем использовать слайдер фотопленки, однако фотографии будут прокручиваться автоматически (а не по нажатию кнопки влево или вправо) и не будет фона самой фотопленки. При этом мы опять не будем использовать сторонние библиотеки, а весь код напишем самостоятельно.

Также мы воспользуемся предыдущим примером и по щелчку на карусели будем загружать в div выбранное пользователем изображение. Для этого мы надлежащим образом задаем событие onclick, а именно вызываем функцию загрузки изображения в div:

```
<div id="scroller_container" onmousemove="_scroller.stop();"
onmouseout="_scroller.timer(_scroller.direct);">
    <div>
        <a href="#1" onclick='imgdiv("s1.jpg")'><img src="s1.
jpg" height="100"></a>
        <a href="#2" onclick='imgdiv("s2.jpg")'><img src="s2.
jpg" height="100"></a>
        <a href="#3" onclick='imgdiv("s3.jpg")'><img src="s3.
jpg" height="100"></a>
        <a href="#4" onclick='imgdiv("s4.jpg")'><img src="s4.
jpg" height="100"></a>
        <a href="#5" onclick='imgdiv("s5.jpg")'><img src="s5.
jpg" height="100"></a>
        <a href="#6" onclick='imgdiv("s6.jpg")'><img src="s6.
jpg" height="100"></a>
        <a href="#7" onclick='imgdiv("s7.jpg")'><img src="s7.
jpg" height="100"></a>
        <a href="#8" onclick='imgdiv("s8.jpg")'><img src="s8.
jpg" height="100"></a>
        <a href="#9" onclick='imgdiv("s9.jpg")'><img src="s9.
jpg" height="100"></a>
    </div>
</div>
```

Полный код примера вместе с комментариями приведен в листинге 16.5.

Листинг 16.5. Карусель фотографий

```
<html>
<head>
<style type="text/css">
    #scroller_container {
        margin: 0 auto;
        width: 530px;
        overflow: hidden;
        line-height:0;
        font-size:0;
    }
    #scroller_container div {
        float: left;
        vertical-align: baseline;
    }
    #scroller_container>div {
        position: relative;
        white-space: nowrap;
        width: 100%;
        padding: 22px 0;
    }
    #scroller_container>div a,
    #scroller_container>div a img {
        padding: 0;
        margin:0;
    }
    #scroller_container>div a img {
        margin: 0 1px;
        border:0;
    }
</style>

<script type="text/javascript">
    var _scroller;
    _scroller = function () { // scroller
        return{
            speed:10, /*скорость, чем больше значние, тем
медленнее движение*/
            direct:-1,/* -1 - движение влево, +1 - вправо*/
            position:0,
            t:null,
            // Инициализация скроллера
            init: function () {
                var el;
                // Установка обработчика колесика мыши
```

```
            el = document.getElementById('scroller_container');
        _scroller.addEvent(el, 'mousewheel', _scroller.wheel);
        _scroller.addEvent(el, 'DOMMouseScroll', _scroller.wheel);

                _scroller.timer(_scroller.direct); //
запускаем скроллер
            },

            // Обработчик колесика мыши
            wheel: function (e) {
                _scroller.stop();
                e = e ? e : window.event;
                /*var wheelElem = e.target ? e.target :
e.srcElement;*/
                var wheelData = e.detail ? e.detail * -1 :
e.wheelDelta / 40;

                // В движке WebKit возвращается значение в 100
раз больше
                if (Math.abs(wheelData) > 100) {
                    wheelData = Math.round(wheelData / 100);
                }
                //_scroller.scroll(wheelData*10);
                _scroller.direct=wheelData>0?1:-1;
                _scroller.timer(_scroller.direct);
                if (window.event) {
                    e.cancelBubble = true;
                    e.returnValue = false;
                    e.cancel = true;
                }
                if (e.stopPropagation && e.preventDefault) {
                    e.stopPropagation();
                    e.preventDefault();
                }
                return false;
            },

            // Функция скроллера
            scroll: function (wheel) {
                var el = document.getElementById('scroller_
container').firstElementChild;
                var o, oi, width;
                _scroller.position += wheel;
                if (wheel>0) {
                    if (_scroller.position >= 0) {
```

```
                    // берем последнюю картинку и вставляем ее
в начало
                    // в этот момент можно подгружать
находящуюся левее картинку
                    // и удалить последнюю
                        o=el;//.firstElementChild; // контейнер
с картинками
   // последняя картинка вместе с анкором
                        oi=o.lastElementChild;
                     // размер картинки
      width=oi.firstElementChild.clientWidth;
                        o.insertBefore(oi,o.firstElementChild);
                        _scroller.position-=width;
                    }
                }
                else {
                    o=el;//.firstElementChild; // контейнер с
картинками
                    oi=o.firstElementChild; // первая картинка
вместе с анкором
                    width=oi.firstElementChild.clientWidth; //
размер
                    if(_scroller.position < -width){
                     // если картинка ушла влево из зоны
видимости
   // переносим ее в конец списка
                     // В этот момент можно подгружать
следующую
                     // картинку и удалить первую
                        o.appendChild(oi);
                        _scroller.position+=width;
                    }
                }
                el.style.left = _scroller.position + 'px';
            },

            // Таймер скроллера
            timer: function (wheel) {
                _scroller.stop();
                _scroller.t = setInterval("_scroller.scroll("
+ wheel + ");", _scroller.speed);
            },

            // Остановка скроллера
            stop: function () {
                if (_scroller.t != null) {
```

```
                clearInterval(_scroller.t);
                _scroller.t = null;
            }
        },

        // назначить обработчик события
        addEvent:function(el, evType, fn, useCapture) {
            if (el.addEventListener) {
                el.addEventListener(evType, fn,
useCapture);
            }else if (el.attachEvent) {
                var r = el.attachEvent('on' + evType, fn);
            }else el['on' + evType] = fn;
        }
    };
}();

window.onload=function(){setTimeout(_scroller.init,100);};
// стартуем с задержкой, чтобы все прогрузилось

// Функция загрузки изображения в div
function imgdiv(url)
{
var el=document.getElementById('img_div');
var img=new Image();
img.onload=function()
    {
    el.style.width=img.width+'px';
    el.style.height=(img.height+20)+'px';
    el.innerHTML='<center><img src='+url+' width="'+img.
width+'" height="'+img.height+'" /><br> ('+img.width+'x'+img.
height+')';
    }
el.innerHTML='Загружается...';
img.src=url;
}
</script>
</head>
<body>
<h1>Карусель фотографий</h1>

<div id="scroller_container" onmousemove="_scroller.stop();"
onmouseout="_scroller.timer(_scroller.direct);">
    <div>
        <a href="#1" onclick='imgdiv("s1.jpg")'><img src="s1.
jpg" height="100"></a>
```

```
        <a href="#2" onclick='imgdiv("s2.jpg")'><img src="s2.
jpg" height="100"></a>
        <a href="#3" onclick='imgdiv("s3.jpg")'><img src="s3.
jpg" height="100"></a>
        <a href="#4" onclick='imgdiv("s4.jpg")'><img src="s4.
jpg" height="100"></a>
        <a href="#5" onclick='imgdiv("s5.jpg")'><img src="s5.
jpg" height="100"></a>
        <a href="#6" onclick='imgdiv("s6.jpg")'><img src="s6.
jpg" height="100"></a>
        <a href="#7" onclick='imgdiv("s7.jpg")'><img src="s7.
jpg" height="100"></a>
        <a href="#8" onclick='imgdiv("s8.jpg")'><img src="s8.
jpg" height="100"></a>
        <a href="#9" onclick='imgdiv("s9.jpg")'><img src="s9.
jpg" height="100"></a>
    </div>
</div>

<div id="img_div" style='BORDER:#9fbddd 1px solid;'></div>
</body>
</html>
```

16.5. Затенение картинки при наведении с помощью фильтра

Опять настало время для простого примера, а именно - сейчас мы реализуем затенение картинки при наведении на нее указателя мыши (лист. 16.6).

Листинг 16.6. Затенение картинки

```
<style>
a img:hover {
filter:alpha(opacity=50); ..-opacity:0.5; opacity:0.5; -khtml-
opacity:0.5;
}
</style>

<script type="text/javascript">
function g(c,w)   // ссылка на объект и признак вкл/выкл
{
if (w==0 && c.style.filter)
    {c.style.filter="light()";
```

```
/*
 применяем фильтр "light", который окрашивает картинку в
определенный цвет...
 и задаем в качестве затемняющего цвета темно-синий оттенок.
 Эта команда используется лишь в том случае, когда браузер
может работать с фильтрами
 - именно поэтому она помещена после оператора if,
проверяющего, существует ли для браузера фильтр light.
*/
    if(c.filters.light)c.filters.light.
addAmbient(150,150,150,180);
// Если нужно вернуть картинке первоначальный вид, то просто
// отменяем все фильтры.
}else   c.style.filter='';
}
</script>
```

Данный сценарий можно использовать для всех картинок на странице, к которым нужно применить эффект "затемнения". В тэг каждой картинки, для которой требуется "затемнение", следует вставить команды "onMouseover="g(this,0)" onMouseout="g(this,1)".

16.6. Галерея Fancy Box

Ранее мы реализовали фото-слайдер и карусель фотографий. Вам ничего не напоминают эти мини-проекты? Правильно, они напоминают фотогалереи, которых в Интернете предостаточно. Как было показано ранее, такую галерею несложно реализовать самостоятельно. Но если вы предпочитаете сразу использовать готовые решения, а не изобретать колесо, можно использовать галерею Fancy Box.

16.6.1. Самая простая галерея

Прежде чем приступить к написанию кода, скачайте плагин FancyBox с официального сайта: **http://fancybox.net/**

Для подключения плагина к вашей странице нужно подключить: библиотеку jQuery, собственно сам плагин FancyBox и таблицу стилей (CSS-файл) FancyBox:

```
<script       src="js/jquery-1.x.x.min.js"       type="text/
javascript"></script>
<script    src="js/jquery.fancybox-1.3.4.js"    type="text/
javascript"></script>
<link  type="text/css"  href="js/jquery.fancybox-1.3.4.css"
rel="stylesheet" />
```

Должен отметить, что FancyBox давно не обновлялся (однако это не говорит о том, что он плохо работает!) и поэтому даже на момент написания этих строк (конец 2016 года) его текущая версия - 1.3.4.

Секция кода, загружающая галерею, будет выглядеть так:

```
<script type="text/javascript">
$(function(){
  $("a").fancybox();
});
</script>
```

После того как галерея подключена, ссылки на рисунки вы можете формировать так:

```
  <a title="Ночной город" href="photos/1.jpg">
    <img src="photos/1_small.jpg" /></a>
```

Атрибут title ссылки (тега <a>) будет служить подписью картинки в галерее, href задает изображение, которое будет загружено. В качестве тела ссылки принято использовать загруженные миниатюры изображений. Я использовал миниатюры размером 120x80 пикселей, а сами изображения – 800 x 600. Изменить размер изображений пакетно (то есть для многих сразу) можно с помощью программы FastStone Image Viewer или любой другой подобной программы (например, ACDSee).

Созданная нами галерея выглядит, как показано на рис. 16.4. Изображение, указанное в атрибуте href ссылки будет загружено в отдельном окне, закрыть которое можно с помощью кнопки X в верхнем правом углу.

Полная версия кода приведена в листинге 16.6.

Листинг 16.6. Страница с галереей

```
<html>
<head>
```

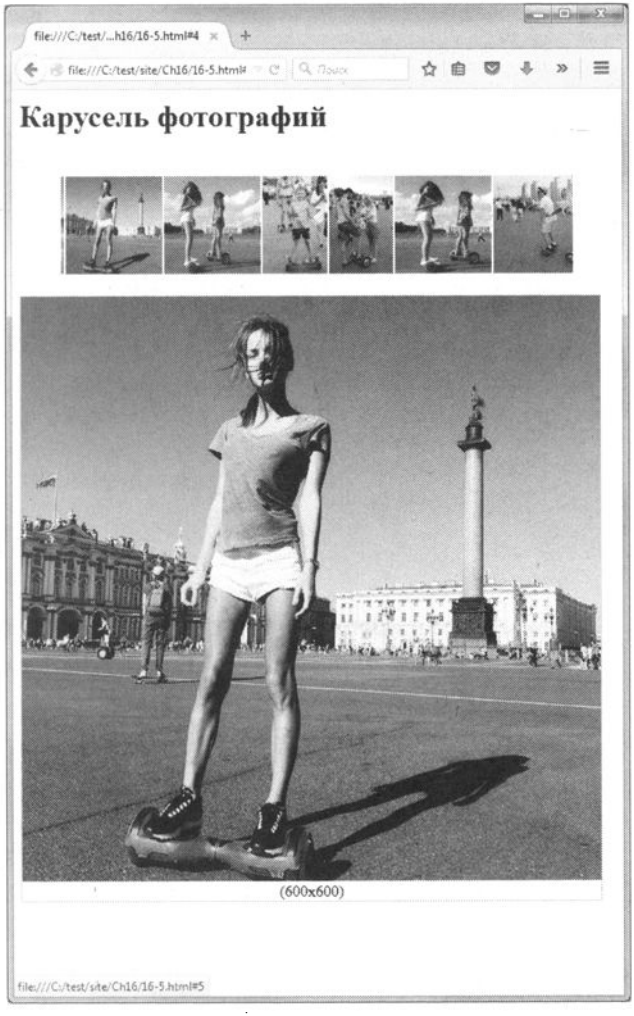


Рис. 16.4. Пример галереи

```
<title>16-6</title>
<script src="js/jquery-1.x.x.min.js" type="text/javascript"></
script>
<script src="js/jquery.fancybox-1.3.4.js" type="text/
javascript"></script>
<link type="text/css" href="js/jquery.fancybox-1.3.4.css"
rel="stylesheet" />
<style type="text/css">
img { padding:2px; border:2px solid #ccc; margin:2px; }
a { outline:none; text-decoration: none; }
```

```
</style>
<script type="text/javascript">
$(function(){
  $("a").fancybox();
});
</script>
</head>
<body>
<p>
  <a title="Ночной город" href="photos/1.jpg">
    <img src="photos/1_small.jpg" /></a>
  <a title="Река" href="photos/2.jpg">
    <img src="photos/2_small.jpg" /></a>
  <a title="Река ночью" href="photos/4.jpg">
    <img src="photos/4_small.jpg" /></a>

</body>
</html>
```

16.6.2. Просмотр предыдущей и следующей фотографии. Изменение внешнего вида окна галереи

У нашей небольшой галереи есть недостаток. Изображение можно только открыть и закрыть, но если есть несколько связанных изображений, пользователь не может просмотреть их, используя кнопки вперед/назад в окне изображения. Ему нужно закрывать окно изображения, выбирать следующую картинку, потом опять закрывать окно и опять открывать следующее изображение. Не очень удобно.

Сейчас мы предоставим возможность просматривать всю галерею в режиме именно галереи, то есть когда пользователь может использовать кнопки "Назад" и "Вперед" для просмотра предыдущей и следующей фотографии. Заодно мы изменим стиль окна, сделав его похожим на окно галереи Lightbox.

Изменений будет немного. Во-первых, ко всем ссылкам нужно добавить атрибут rel="group":

```
<a rel="group"  title="Река" href="photos/5.jpg">
    <img src="photos/5_small.jpg" /></a>
  <a rel="group" title="Река ночью" href="photos/3.jpg">
    <img src="photos/3_small.jpg" /></a>
```

Во-вторых, нужно изменить сам JS-код вызова галереи:

```
<script type="text/javascript">
$(function(){
   function formatTitle(title, currentArray, currentIndex,
currentOpts) {
    return '<div id="tip-title"><span>
                <a href="javascript:;" onclick="$.fancybox.
close();">
            <img src="img/closelabel.gif" /></a></span>' +
   (title && title.length ? '<strong>' + title + '</strong>'
: '' ) +
   'Photo ' + (currentIndex + 1) + ' from ' + currentArray.
length +
           '</div>';
  }
  $("a[rel='group']").fancybox({
    "showCloseButton": false,
      "titlePosition" : "inside",
      "titleFormat": formatTitle
  });
});
</script>
```

Мы используем функцию formatTitle() для форматирования заголовка, по сути, задаем новый вид окна. Наш формат использует новые стили, которые нужно тоже определить на нашей странице:

```
#tip-title { text-align:left; }
#tip-title strong { display:block; margin-right:80px; }
#tip-title span { float:right; }
#tip-title img { border:none; }
```

Вот и все, собираем все это вместе (лист. 16.7) и открываем для просмотра. Как видите, оформление окна просмотра стало совсем другим (оно стало похожим на галерею LightBox), а также появились кнопки "Вперед"/"Назад" (на рисунке видна только кнопка "Вперед").

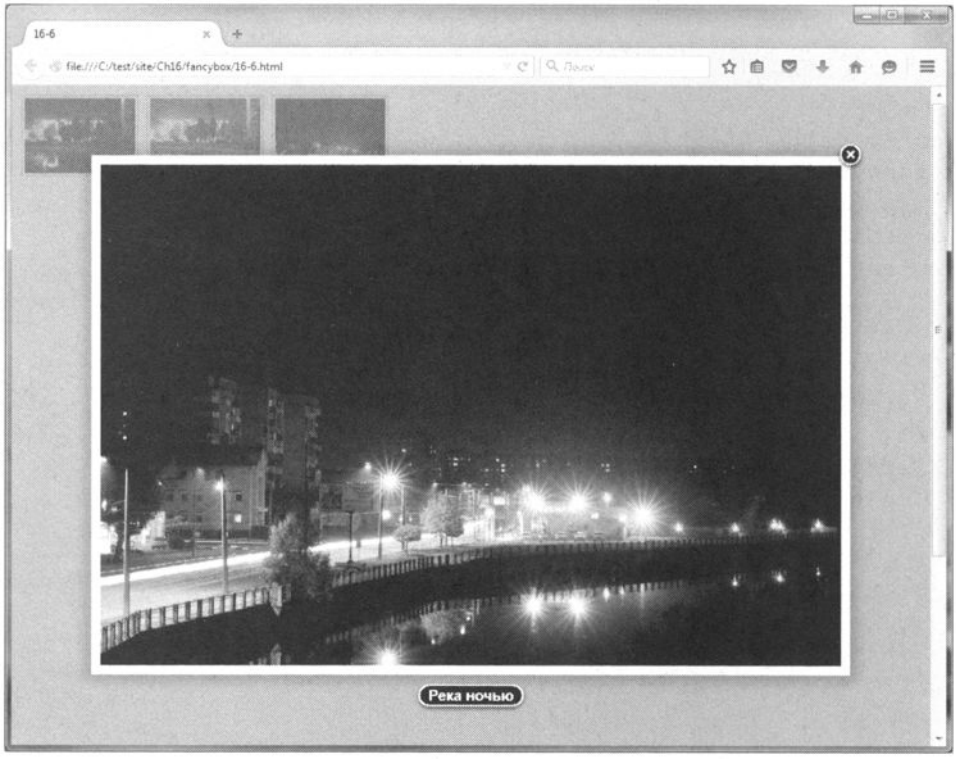


Рис. 16.5. Видоизмененная галерея: при подведении указателя мыши появляется стрелка (в данном случае влево)

Листинг 16.7. Второй вариант галереи

```
<html>
<head>
<title>16-7</title>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<script src="js/jquery-1.5.2.min.js" type="text/javascript"></
script>
<script src="js/jquery.fancybox-1.3.4.js" type="text/
javascript"></script>
<link type="text/css" href="js/jquery.fancybox-1.3.4.css"
rel="stylesheet" />
<style type="text/css">
img { padding:2px; border:2px solid #ccc; margin:2px; }
a { outline:none; text-decoration: none; }
```

```
#tip-title { text-align:left; }
#tip-title strong { display:block; margin-right:80px; }
#tip-title span { float:right; }
#tip-title img { border:none; }
</style>
<script type="text/javascript">
$(function(){
  function formatTitle(title, currentArray, currentIndex,
currentOpts) {
    return '<div id="tip-title"><span><a href="javascript:;"
onclick="$.fancybox.close();"><img src="img/closelabel.gif"
/></a></span>' + (title && title.length ? '<strong>' + title +
'</strong>' : '' ) + 'Photo ' + (currentIndex + 1) + ' from '
+ currentArray.length + '</div>';
  }
  $("a[rel='group']").fancybox({
    "showCloseButton": false,
   "titlePosition" : "inside",
   "titleFormat": formatTitle
  });
});
</script>
</head>
<body>
<p>
  <a rel="group"  title="Река" href="photos/5.jpg">
    <img src="photos/5_small.jpg" /></a>
  <a rel="group" title="Река" href="photos/3.jpg">
    <img src="photos/3_small.jpg"  /></a>
  <a rel="group" title="Театр" href="photos/6.jpg">
    <img src="photos/6_small.jpg"  /></a>
</p>
</body>
</html>
```

Если же вам не нужен внешний вид в стиле LightBox, но вы хотите просматривать изображения в группе (чтобы была возможность перехода к предыдущему и следующему изображению), тогда все еще проще. Как уже отмечалось, для всех изображений группы нужно задать атрибут rel="group" в теге <a>. А вот JavaScript-код придется изменить так:

```
<script type="text/javascript">
$(function(){
  $("a[rel^='group']").fancybox({
```

```
      "cyclic": true,
    "speedIn": 1000,
    "speedOut": 1000,
    "overlayColor": "#ccc",
    "overlayOpacity": 0.6,
    "transitionIn": "elastic",
    "transitionOut": "elastic"
  });
});
</script>
```

Здесь мы определяем атрибут группы, а именно rel="group", фактически можно было бы использовать и другой атрибут, например title, id, но этот атрибут не отображается браузером и реально ни на что не влияет, поэтому рекомендуется использовать именно его.

Параметр cyclic (если он равен true) означает, что изображения в галерее будут выводиться замкнуто, то есть по кругу и после просмотра последнего изображения будет показано первое.

Параметры speedIn и speedOut задает время выполнения анимации и открытия и закрытия полноразмерного изображения. По умолчанию - 300 мс, но мы увеличили это время до 1000 мс (1 секунда).

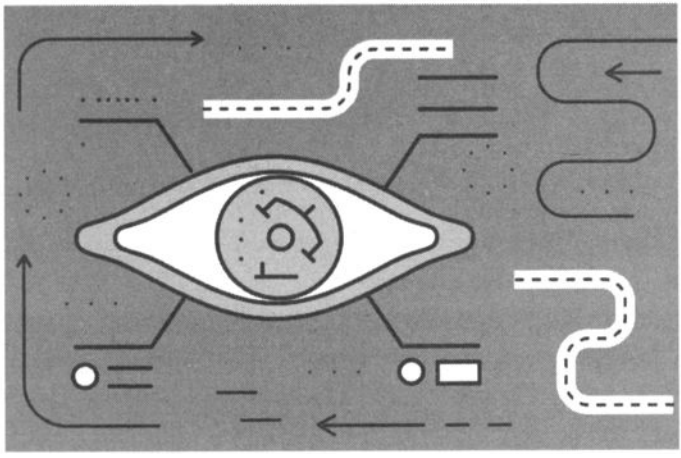
Параметр overlayColor задает цвет слоя, который будет накладываться на основную страницу. По умолчанию используется значение #666, но мы переопределили это дьявольское значение на #ccc.

Параметр overlayOpacity задает прозрачность дополнительного слоя, значения принимаются от 0 до 1 с шагом 0.1. По умолчанию используется значение 0.3, но мы сделали слой немного прозрачнее и установили значение 0.6.

Параметры transitionIn и transitionOut задают вид анимационных эффектов при открытии/закрытии изображений. По умолчанию используется значение 'fade', мы же задали значение 'elastic'. Чтобы отказаться от анимационных эффектов, установите значение "none".

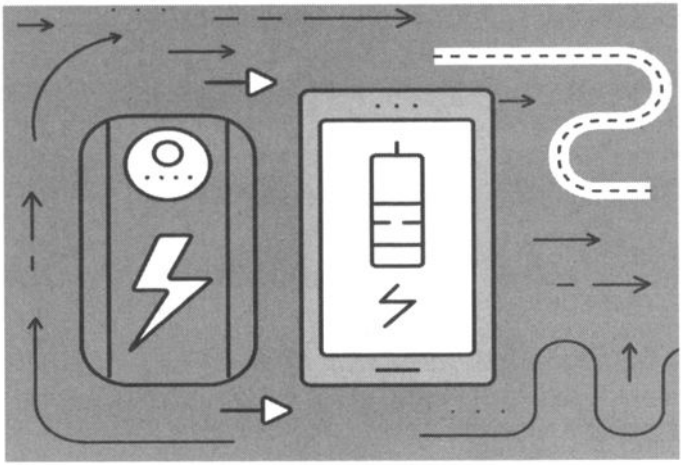
С остальными параметрами плагина вы можете ознакомиться на сайте разработчиков:

http://fancybox.net/api



Глава 17.

Всякие полезности



17.1. Счетчик посещений с помощью Cookies

Сейчас попытаемся создать счетчик посещений с помощью Cookies. Что такое Cookies, объяснять не станем - об этом полно информации в Интернете. Грубо говоря, это специальная область памяти в браузере, которая может хранить различные пользовательские значения. Мы будем использовать Cookies для хранения количества посещений пользователем сайта и даты последнего визита. Имейте в виду, что при очистке Cookies эта информация будет стерта, поэтому там нельзя хранить какую-либо важную информацию.

Для работы с Cookies мы напишем три функции:

- getCookie() - возвращает значение Cookie по имени.
- setCookies() - устанавливает (записывает) значение Cookie;
- parseCookie() - разделяет строку с Cookie на пары имя-значения, выполняет декодирование и добавляет пары в массив с Cookies, который потом легко использовать.

Если Cookies не установлены, то мы выводим строку "Добро пожаловать на наш сайт!"

```
    if ("" == document.cookie)
     { // Инициализация cookie.
      setCookie(1);
     document.write("<div align=right>Добро пожаловать на наш
сайт!.</div>");
     }
```

В противном случае мы запускаем сначала парсинг Cookies, затем выводим информацию о количестве визитов и дате последнего визита, увеличиваем счетчик посещений и сохраняем новое (увеличенное) значение в Cookies:

```
else {
       var cookies = parseCookie();
       if (isNaN(cookies.visits)) cookies.visits = 1;
```

```
        // Вывод приветствия, числа посещений и увеличение
числа посещений на 1.

       document.write("<div align=right><br><i>Мы снова рады
видеть Вас! Вы были у нас - " +
          cookies.visits++ + " раз(а)</i></div>");
       // Вывод даты последнего посещения.
       document.write ("<div align=right><i>Последний раз Вы
были у нас: " + cookies.LastVisit + ".</i></div>");
       // Обновление cookie.
       setCookie(isNaN(cookies.visits)?1:cookies.visits);
    }
```

Полный код файла cookies.js приведен в листинге 17.1.

Листинг 17.1. Код файла cookies.js

```
// возвращает по имени значение, здесь не используется
function getCookie(byname){
    byname=byname+"=";
    nlen = byname.length;
    fromN = document.cookie.indexOf(byname)+0;
    if((fromN) != -1)
        {
         fromN +=nlen
         toN=document.cookie.indexOf(";",fromN)+0;
         if(toN == -1) {toN=document.cookie.length;}
         return unescape(document.cookie.
substring(fromN,toN));
        }
    return null;
   }

 // Разделение cookie
 function parseCookie()
   { var cookieList = document.cookie.split("; ");
   // Массив для каждого cookie в cookieList
   var cookieArray = new Array();
   for (var i = 0; i < cookieList.length; i++) {
       // Разделение пар имя-значение.
       var name = cookieList[i].split("=");
       // Декодирование и добавление в cookie-массив.
       cookieArray[unescape(name[0])] = unescape(name[1]);
    }
   return cookieArray;
```

```
  }

 function setCookie(visits) {
    /* Счетчик числа посещений с указанием даты последнего
посещения
       и определением срока хранения в 1 год. */
    var expireDate = new Date();
    var today = new Date();
    // Установка даты истечения срока хранения.
    expireDate.setDate(365 + expireDate.getDate());
    // Сохранение числа посещений.
    document.cookie = "visits=" + visits +
                      "; expires=" + expireDate.toGMTString()
+ ";";
    // Сохранение настоящей даты как времени последнего
посещения.
    document.cookie = "LastVisit=" + escape(today.
toGMTString()) +
                       "; expires=" + expireDate.toGMTString()
+ ";";
    }

    if ("" == document.cookie)
    { // Инициализация cookie.
     setCookie(1);
     document.write("<div align=right>Добро пожаловать на наш
сайт!.</div>");
    }
    else {
       var cookies = parseCookie();
       if (isNaN(cookies.visits)) cookies.visits = 1;
       // Вывод приветствия, числа посещений и увеличение
числа
       // посещений на 1.

       document.write("<div align=right>
                 <br><i>Мы снова рады видеть Вас! Вы были у
нас - " +
                 cookies.visits++ + " раз(а)</i></div>");
       // Вывод даты последнего посещения.
       document.write ("<div align=right>
                 <i>Последний раз Вы были у нас: " +
                 cookies.LastVisit + ".</i></div>");
       // Обновление cookie.
       setCookie(isNaN(cookies.visits)?1:cookies.visits);
    }
```

Использовать сценарий очень просто. Добавьте следующую строку в место отображения счетчика:

```
<script src="cookies.js"></script>
```

Например:

```
<body>
   <div id="header">
      <h1>Гироскутеры на любой вкус</h1>
         <script src="cookies.js"></script>
      <ul id="navigation">
```

Результат изображен на рис. 17.1. Дату последнего визита вы вряд ли будете выводить, а вот счетчик посещений конкретного пользователя может пригодиться.

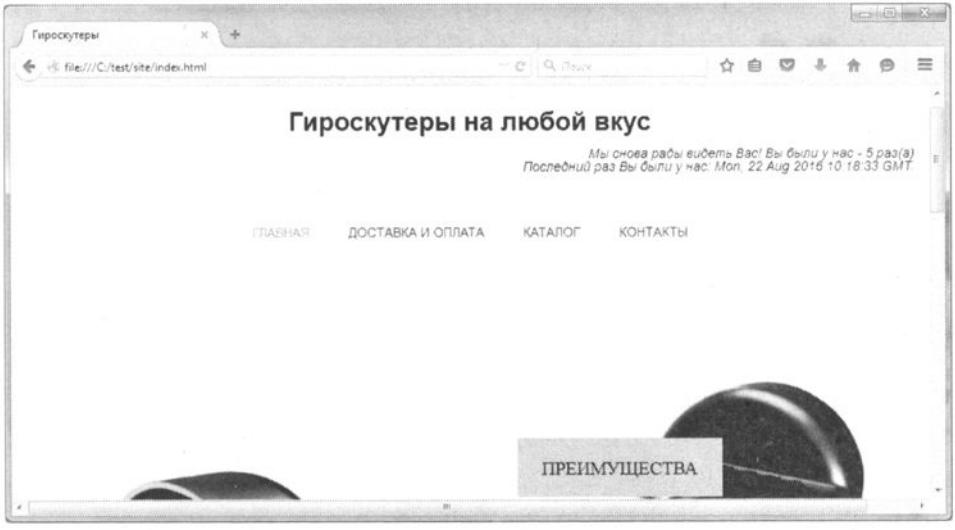


Рис. 17.1. Счетчик в действии

При желании можно оформить счетчик с помощью стилей:

```
.counter {
  position:absolute; display:block;
   padding:5px;
```

```
   border:solid #089dcb;
   border-width:2px;
   border-radius:10px 10px 10px 0px;
   -webkit-border-radius:10px 10px 10px 0px;
   -moz-border-radius:10px 10px 10px 0px;
   -webkit-box-shadow:1px 1px 2px #888888;
   -moz-box-shadow:1px 1px 2px #888888;
   box-shadow:1px 1px 2px #888888;
   background-color:#ffD;
}
```

Потом не забудьте определить этот стиль в HTML-коде:

```
<div class="counter">
   <script src="cookies.js"></script>
</div>
```

Внешний вид такого счетчика показан на рис. 17.2. Как видно из рисунка, вывод даты последнего визита отключен (путем комментирования соответствующей строчки в сценарии).

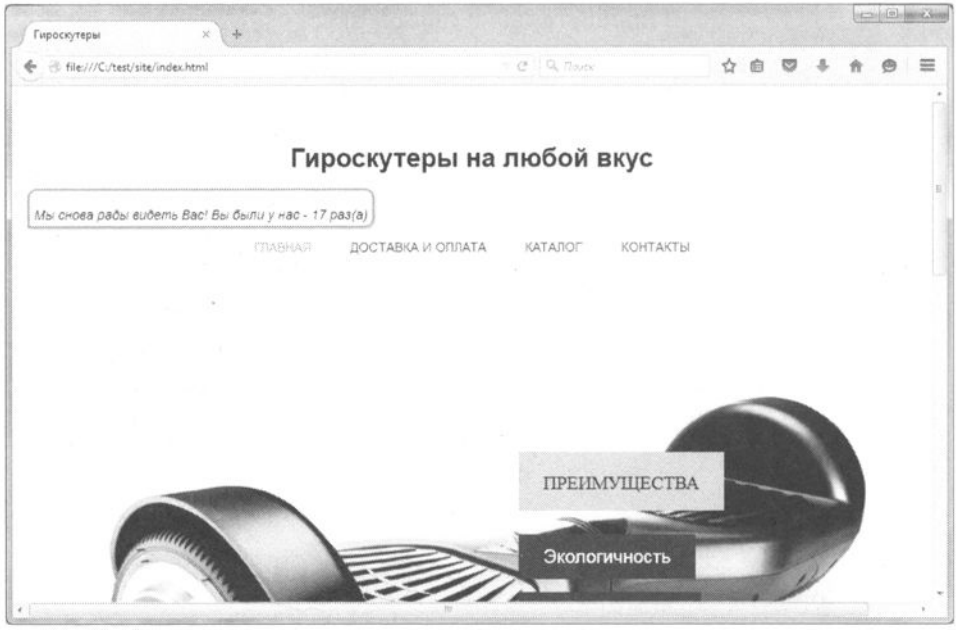


Рис. 17.2. Изменен внешний вид счетчика (с помощью CSS)

17.2. Запрещаем браузеру выделять текст

Сейчас мы поговорим о защите авторских прав техническими средствами, а именно рассмотрим, как запретить браузеру выделять важный текст на странице - пользователь не сможет его выделить, значит не сможет копировать. Конечно, он в состоянии открыть исходный код страницы и потом уже просмотреть его, но далее будет показано, как запретить просмотр HTML-кода.

Принцип следующий: в HTML-код вы добавляете блок текста с id "noselect".

```
<div id="noselect">
Описание товара
</div>
```

Далее вам нужно добавить следующий сценарий в конец страницы (лист. 17.2):

Листинг 17.2. Запрет копирования текста

```
<script type="text/javascript">
// Функция отключает выделение заданного фрагмента страницы
function disableSelection(target){
if (typeof target.onselectstart!="undefined")
    target.onselectstart=function(){return false}
else if (typeof target.style.MozUserSelect!="undefined")
    target.style.MozUserSelect="none"
else
    target.onmousedown=function(){return false}
target.style.cursor = "default"
}

if (document.getElementById("noselect"))
   disableSelection(document.getElementById("noselect"));

</script>
```

Данный код запрещает выделение текста для элемента с именем "noselect".

17.3. Добавляем информацию об авторских правах

Есть еще один метод борьбы с пиратами. При каждом копировании текста добавлять к выделенному "пиратом" тексту информацию об авторских правах. Конечно, ее легко удалить, но это придется делать для каждого скопированного фрагмента, что очень неудобно. Возможно, вы попортите больше крови этим трюком, чем даже предыдущим. Код приведен в листинге 17.3.

Листинг 17.3. Добавляем информацию об авторских правах к скопированному тексту

```
<script type="text/javascript">
function addLink() {
    var body_element = document.getElementsByTagName('body')
[0];
    var selection;
    selection = window.getSelection();
    var pagelink = "<br /><br />
Скопировано с
<a href='"+document.location.href+"'>"
+document.location.href+"</a>
<br />© dkws.org.ua";

    var copytext = selection + pagelink;
    var newdiv = document.createElement('div');
    newdiv.style.position='absolute';
    newdiv.style.left='-99999px';
    body_element.appendChild(newdiv);
    newdiv.innerHTML = copytext;
    selection.selectAllChildren(newdiv);
    window.setTimeout(function() {
        body_element.removeChild(newdiv);
    },0);
}
document.oncopy = addLink;
</script>
```

17.4. Запрет просмотра HTML-кода

Еще один трюк - можно запретить просматривать HTML-код страницы. Добавив на страницу код, указанный в листинге 17.4, вы отключите контекст-

ное меню, которое появляется при нажатии правой кнопки мыши. Именно в этом меню есть команда просмотра HTML-кода. Решение так себе, но для начинающих "пиратов" сойдет. А более продвинутые скачают ваш сайт посредством программ вроде Teleport и все равно смогут просмотреть код любой страницы в их любимом текстовом редакторе.

Листинг 17.4. прет просмотра HTML-кода

```
<!--[if gte IE 5]><script type="text/javascript">
 createPopup().show( 0, 0, 0, 0, 0 );
 </script><![endif]-->
 <body oncontextmenu="return false;">
```

Данный код работает в IE, Firefox и Chrome. В остальных браузерах он не проверялся.

17.5. Отложенная загрузка файла

Загружая файлы с различных файлообменников, вы наверняка видели, что загрузка начинается не сразу, а спустя некоторое время, на протяжении которого вы любуетесь рекламой. Сейчас попробуем организовать такой же алгоритм. Конечно, для нашего сайта он не нужен, но пример довольно интересный, возможно, он пригодится вам, когда будете работать над другим проектом.

Код полностью рабочий и вы можете его использовать, заменив только путь файла для загрузки и код рекламного баннера (лист. 17.5).

Листинг 17.5. Отложенная загрузка файла

```
<p>В течение нескольких секунд начнется загрузка файла.<br>
    Если этого не произошло, <a href="/download.zip">нажмите
сюда</a>.</p>
<div id="download_div" style="position:absolute; top:-
100px"></div>
<script type="text/javascript">
setTimeout(function() {
    document.getElementById('download_div').innerHTML=
        '<iframe src="/download.zip" style="width:1px;
height:1px"></iframe>';
}, 10000); // 10 секунд
</script>
```

```
<br /> <br /> <br />
<h2>Здесь может быть ваша реклама</h2>
```

17.6. Всплывающие окна

В этом разделе будет показано, как сделать всплывающее окно при входе на сайт и при уходе с сайта. Учтите, что в большинстве случаев браузеры будут блокировать всплывающие окна, и эта тактика не всегда приветствуется пользователями. В листинге 17.6 показано, как сделать всплывающее окно при входе на сайт.

Листинг 17.6. Всплывающее окно при входе на сайт

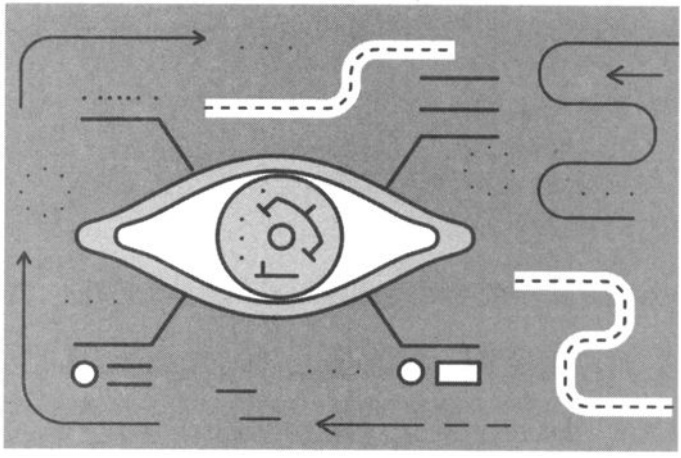
```
<script type="text/javascript">
<!--
browserVer = parseInt(navigator.appVersion);

Top=window.open("http://example.com/",'exaple', 'toolbar=1,
location=1, status=1, menubar=1, scrollbars=1, resizable=1');

if( browserVer >= 4) {
    window.focus() ;
}
// -->
</script>
```

А в листинге 17.7 приведен код всплывающего окна при уходе с сайта.

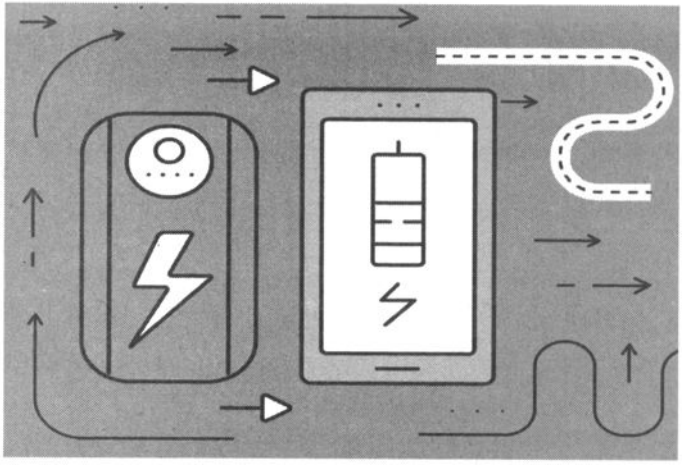
Листинг 17.7. Код всплывающего окна при уходе с сайта

```
<body onunload="ex()">
...
<script type="text/javascript">
<!--
function ex()
{
    open("http://example.com/",'example','toolbar=1,locati
on=1,
    status=1,menubar=1,scrollbars=1,resizable=1');
    window.focus() ;
}
// -->
</script>
```

Здесь мы устанавливаем обработчик события onunload - функцию ex(), которая и открывает всплывающее окно.



Заключение



В качестве заключения хотелось бы подвести итог проделанной работе. Итак, мы начали с обычного HTML-сайта, не содержащего каких-либо скриптов, только CSS-код для оформления.

Первым делом мы добавили к нашему сайту меню, слайдер с продуктами и всплывающие подсказки. Меню и слайдер изображены на рис. 1. О том, как создать слайдер, было написано в главе 5, всплывающие подсказки рассмотрены в главе 6, а меню - в главе 9.

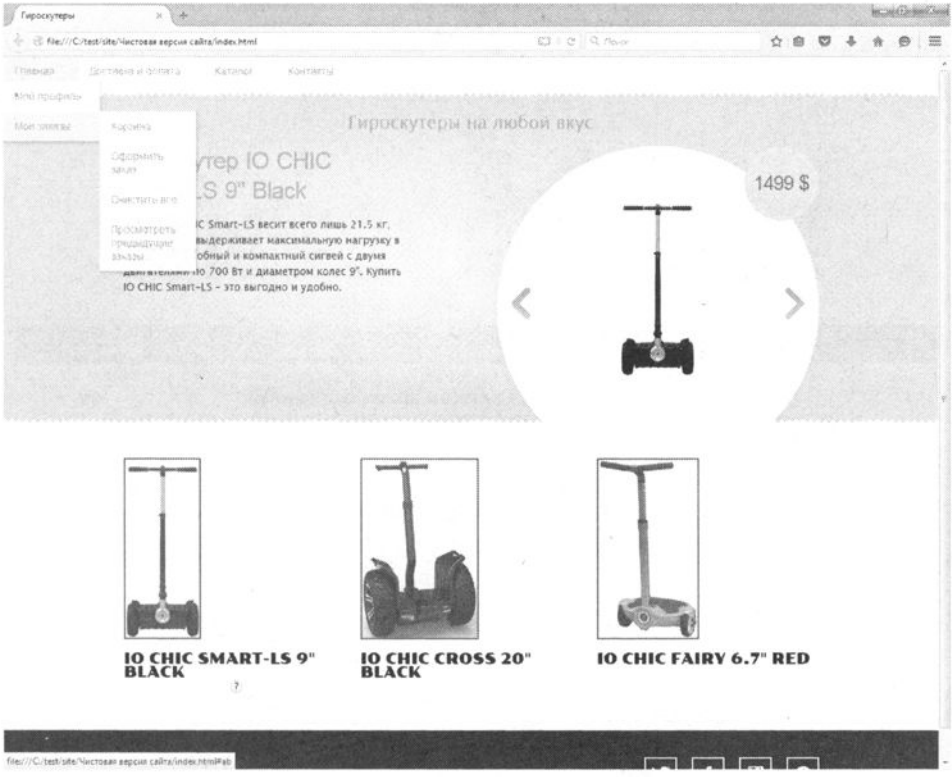


Рис. 1. Меню и слайдер

В главе 15 было показано, как создавать диалоги, которые можно перетаскивать мышкой, а в главе 17 мы создали персональный счетчик пользователя на базе Cookies. Объединение полученных знаний показано на рис. 2: диалог с количеством посещений.

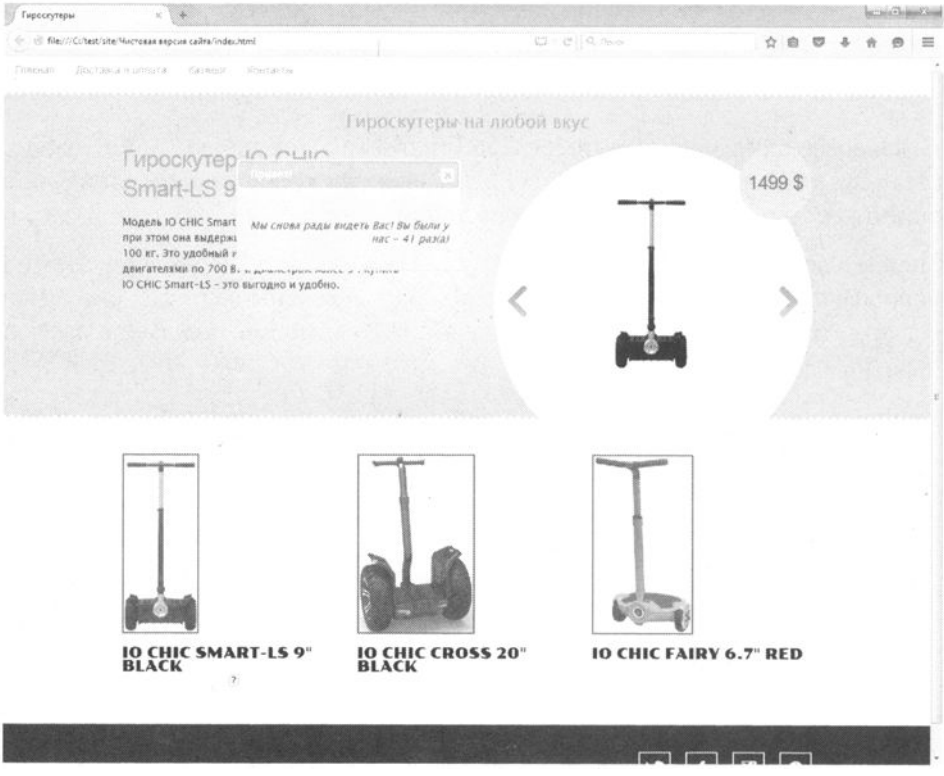


Рис. 2. Диалог с количеством посещений

Рис. 3. Страница товара

Рис. 4. Форма заказа

Мы значительно улучшили страницу товара (рис. 3, 4). Во-первых, мы использовали jQuery UI для организации вкладок товара. Можно было бы использовать и виджет Accordion (см. гл. 15) - это как вам больше нравится. Также мы украсили подвал сайта слайдером в виде фотопленки, что сделало сайт немного живее. При желании вы можете использовать и другие галереи, разработанные нами в главе 16.

Для нашей формы заказа мы добавили проверку полей, а также проверку e-mail, чтобы убедиться, что введенный пользователем электронный адрес верный (см. гл. 11). При желании можно было бы на странице товара использовать галерею Fancy Box, которую мы рассмотрели в главе 16, но я решил не перегружать страницу скриптами - при желании вы можете это сделать самостоятельно.

Также нами в главе 13 была разработана эффектная лендинг-страница, которая при правильном использовании может повысить продажи и привлечь аудиторию (рис. 5).

Рис. 5. Лендинг-страница

В конечном итоге наш сайт стал напоминать современный сайт по продаже товара. Конечно, некоторые функции просто не под силу JavaScript и все равно придется привлекать PHP (например, для обработки форм, для вывода товара из каталога, если не хотите делать это вручную и т.д.). Однако средствами JavaScript можно существенно усовершенствовать интерфейс пользователя, что и было сделано.

В книге приводится много советов и трюков, но не стоит переусердствовать и применять все из них в вашем сайте. Не перегружайте страницу скриптами, и пользователи будут вам благодарны! Выберите только то, что действительно необходимо для вашей страницы и вашим пользователям.

Теперь о примерах:

- Результирующая версия сайта: **http://www.nit.com.ru**
- Только слайдер: **http://www.nit.com.ru**
- Лендинг страница (гл. 13): **http://www.nit.com.ru**
- Примеры использования jQuery UI: **http://www.nit.com.ru**

- Примеры для главы 16 (изображения): **http://www.nit.com.ru**

Остальные примеры довольно просто воспроизвести самостоятельно.

Группа подготовки издания:

Зав. редакцией компьютерной литературы: *М. В. Финков*

Редактор: *Е. В. Финков*

Корректор: *А. В. Громова*

12+

ООО «Наука и Техника»

Лицензия №000350 от 23 декабря 1999 года.

192029, г. Санкт-Петербург, пр. Обуховской обороны, д. 107.

Подписано в печать 02.07.2018. Формат 70x100 1/16.

Бумага газетная. Печать офсетная. Объем 17 п. л.

Тираж 2000. Заказ 6267.

Отпечатано с готовых файлов заказчика
в АО «Первая Образцовая типография»,
филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ»
432980, г. Ульяновск, ул. Гончарова, 14

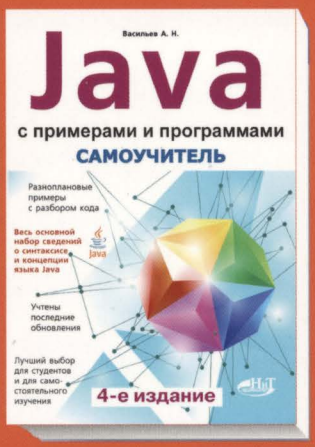
А. П. Никольский

JavaScript
НА ПРИМЕРАХ

ПРАКТИКА, ПРАКТИКА И ТОЛЬКО ПРАКТИКА

Эта книга является превосходным учебным пособием для изучения языка программирования JavaScript на примерах. Изложение ведется последовательно: от написания первой программы до создания полноценных проектов - интерактивных элементов (типа слайдера, диалоговых окон) интернет-магазина, лендинговой страницы и прочего. По ходу даются все необходимые пояснения и комментарии. Книга написана простым и доступным языком. Лучший выбор для результативного изучения JavaScript!

Издательство "Наука и Техника"



ISBN 978-5-94387-762-9

978-5-94387-7629

Издательство "Наука и Техника"
г. Санкт-Петербург

Для заказа книг:
(812) 412-70-26
e-mail: nitmail@nit.com.ru
www.nit.com.ru