JavaScript

НАРОДНЫЕ СОВЕТЫ

ВЛАДИМИР ДРОНОВ

ОБЩИЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ

РАБОТА С WEB-ОБОЗРЕВАТЕЛЕМ

РАБОТА С WEB-СТРАНИЦАМИ И ИХ ЭЛЕМЕНТАМИ

РАБОТА С ДАННЫМИ

СЛОЖНОЕ WEB-ПРОГРАММИРОВАНИЕ

bhv

Владимир Дронов

JAVASCRIPT
НАРОДНЫЕ СОВЕТЫ

Санкт-Петербург
«БХВ-Петербург»
2007

УДК 681.3.06
ББК 32.973.26-018.1
Д75

Дронов В. А.

Д75 JavaScript. Народные советы. — СПб.: БХВ-Петербург, 2007. — 464 с.: ил.

ISBN 978-5-94157-961-7

Книга представляет собой подборку решений, зачастую неочевидных, типичных проблем Web-программирования, приемов, советов и готовых Web-сценариев. Рассмотрены следующие темы: полезные функции и объекты языка JavaScript, написание сценариев и обработка событий, получение сведений о Web-обозревателе и управление им, манипуляции и эффекты с Web-страницами и их элементами, работа с графикой, гиперссылками и полосами навигации, вывод информации о таблицах, эффекты с фреймами, управление свободно позиционируемыми контейнерами, создание мультимедийных элементов и управление ими, простейший ввод-вывод, сохранение и передача данных, работа с формами и элементами управления, простейшие и более сложные приемы Web-программирования, отладка Web-приложений и др.

Для Web-дизайнеров и Web-программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 20.10.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 37,4.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-94157-961-7

Оглавление

Введение

Язык JavaScript сейчас популярен как никогда. Сложные Web-страницы, реагирующие на действия посетителя, заполонили Интернет. Возникла профессия Web-программиста, специалиста по написанию *Web-сценариев* (или просто *сценариев*) — программ на языке JavaScript, встраиваемых в HTML-код страниц и собственно дающих им интерактивность. А на горизонте маячат новые технологии Web-программирования, дающие посетителям сайтов новые, невиданные доселе возможности.

И, разумеется, нет недостатка в книгах по JavaScript и Web-программированию. Толстые и тонкие, поверхностные и подробные, умные и бестолковые — они издаются и переиздаются в изрядных количествах. В любом более-менее приличном книжном магазине можно найти сразу несколько наименований таких книг — на любой вкус и любую потребность.

И вот еще одна книга. Все по тому же JavaScript. Стоило ли из-за нее переводить бумагу?

Чего не дают обычные руководства?

Большинство издаваемых ныне книг по JavaScript представляют собой либо руководства для начинающих, либо справочники. Справочники мы сразу исключим, так как, во-первых, они немногочисленны (увы!..), а во-вторых, рассчитаны все-таки на подготовленных читателей. Возьмем с полки магазина любое руководство и, пока продавец не возмутился: "Здесь вам не читальный зал!" — полистаем.

Итак, руководство по JavaScript. Описание языка плюс минимальные справочные сведения плюс кое-какие примеры сценариев, совсем несложных. Прочитав все это, желающий стать Web-программистом узнает сам язык, напишет несколько собственных сценариев и, возможно, продолжит свое обу-

чение по другим книгам или материалам, выложенным в Интернете (благо их сейчас хватает). И, разумеется, начнет писать свои сценарии.

И тут-то у него появятся первые проблемы. Дело в том, что зачастую в таких руководствах не рассказывается, как сделать ту или иную вещь. Описан, скажем, объект `String` (объектное представление строкового значения), но не рассказано, как выполнить замену всех вхождений заданной подстроки другой подстрокой. И приходится свежеиспеченному Web-программисту, едва научившемуся писать свои первые выражения JavaScript, придумывать, как это сделать, либо искать уже готовый сценарий.

Итак, главная проблема большинства руководств — они рассказывают, что можно сделать, но не рассказывают, как сделать ту или иную вещь. Можно сказать, что они дают россыпь отдельных деталей, но не прилагают к ним инструкцию по сборке.

Собственно, не стоит этого от них требовать — у руководств весьма специфические задачи. Здесь нужны совсем другие книги, этакие сборники советов.

Народные советы

А ведь тот же самый сценарий по замене одной подстроки другой уже наверняка кем-то написан! (В конце концов, задача эта типична для Web-программиста, имеющего дело, в основном, со строковыми данными.) А значит, писать его во второй (десятый, сотый, тысячный) раз не имеет смысла. Нужно только обратиться к программирующей общественности, к программирующему народу — и он обязательно поможет!

Так вот в этой книге как раз и приведены решения типичных задач Web-программирования и типичные Web-сценарии, которые можно сразу же вставить в Web-страницу. Этакая сокровищница знаний программирующего народа. Народные советы, как гласит само название этой книги.

> ***Народ замечает***
>
> Да-да, замечу без ложной скромности, это так. Правда, тут и сам автор книги здóрово постарался, но тоже не без моей помощи.

Итак, что же мы найдем в этой книге?..

- Решения типичных проблем Web-программирования.
- Типичные Web-сценарии, доступные для использования на Web-страницах без всяких ограничений.
- Собственно народные советы (примечания с заголовком "Народ советует").

- Замечания народа по различным вопросам ("Народ замечает").
- Предупреждения народа о возможных проблемах ("Народ предупреждает!").

А вот чего в этой книге даже не стоит искать, так это руководств по HTML, CSS, JavaScript и Web-обозревателям. Для этого существуют другие книги.

Все это работает?

Еще как! Проверено самим автором на следующих Web-обозревателях:

- Microsoft Internet Explorer 6.0 SP2 русский;
- Opera 8.54 русский;
- Mozilla Firefox 1.5.0.3 русский;
- Mozilla 1.7;
- Netscape Navigator 7.0 Preview Release 1.

Практически все Web-сценарии, приведенные в этой книге, тестировались только на первых трех Web-обозревателях как самых популярных на данный момент. Только сценарий, приведенный в листинге 3.1 (функция `jspsGetProgramInfo`, выдающая сведения о Web-обозревателе), проверялся на всех пяти программах.

Большинство приведенных в этой книге решений работает на всех Web-обозревателях (точнее, на первых трех: Internet Explorer, Opera и Firefox). Если же какое-то решение функционирует только на определенных Web-обозревателях, их названия будут указаны в скобках (например, "Решение (Opera)" для решения, работающего только на Opera).

Типографские соглашения

В этой книге часто приводятся форматы использования различных свойств и вызова различных функций и методов. Нам необходимо выучить типографские соглашения, используемые для их написания.

> ***Народ предупреждает!***
>
> Все эти типографские соглашения применяются автором только при описании формата использования свойств и вызова функций и методов. В коде примеров они не имеют смысла.

Так, в угловые скобки (< >) заключаются названия параметров или фрагментов кода, которые, в свою очередь, набраны курсивом. В код реального сце-

нария, разумеется, должен быть подставлен реальный параметр или реальный код. Например:

```
document.createElement(<Имя тега>);
```

Здесь вместо подстроки *<Имя тега>* должно быть подставлено реальное имя тега.

В квадратные скобки ([]) заключаются необязательные фрагменты кода. Например:

```
AnimatedElement(<Элемент страницы>, <Интервал>, [<Массив выражений>]);
```

Здесь *Массив выражений* может присутствовать, а может и отсутствовать.

Если в какое-либо место сценария должна быть подставлена команда, выбираемая из некоего ограниченного набора, в этом месте приводятся все команды данного набора, разделенные символом | ("вертикальная черта"). Например:

```
<PUBLIC:ATTACH EVENT="oncontentready|ondocumentready"
```

Здесь в качестве значения атрибута EVENT тега PUBLIC:ATTACH может присутствовать либо "oncontentready", либо "ondocumentready" (но не оба одновременно).

Слишком длинные, не помещающиеся на одной строке выражения JavaScript автор разрывает на несколько строк и в местах разрывов ставит знаки ↳. Например:

```
<PUBLIC:ATTACH EVENT="oncontentready|ondocumentready"
↳[FOR="<Элемент страницы>"]
↳ONEVENT="<Вызов функции-обработчика события>"/>
```

Приведенный выше код разбит на три строки, но должен быть набран в одну. Знаки ↳ при этом должны быть удалены.

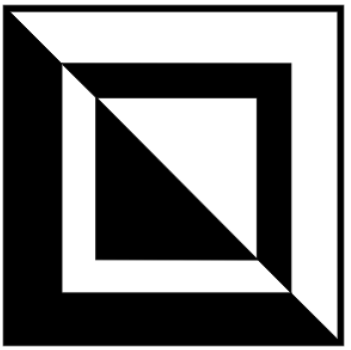
> ***Народ предупреждает в последний раз!***
>
> Все приведенные выше типографские соглашения имеют смысл только при описании формата использования свойств и вызова функций и методов. В коде примеров используется только знак ↳.

Что ж, автор вроде все сказал... Теперь пусть за него говорит всезнающий народ! Предоставим ему слово!..

Благодарности

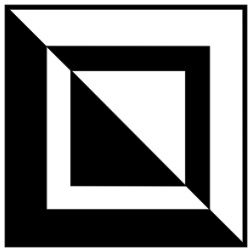
Автор приносит благодарности своим родителям, знакомым и коллегам по работе.

- Губине Наталье Анатольевне, начальнику отдела АСУ Волжского гуманитарного института (г. Волжский Волгоградской обл.), где работает автор, — за понимание и поддержку.
- Всем работникам отдела АСУ — за понимание и поддержку.
- Родителям — за терпение, понимание и поддержку.
- Архангельскому Дмитрию Борисовичу — за дружеское участие.
- Шапошникову Игорю Владимировичу — за содействие.
- Евгению из Волгограда — за фильмы ужасов — лучшее средство для развития чувства юмора.
- Рыбакову Евгению Евгеньевичу, заместителю главного редактора издательства "БХВ-Петербург", — за неоднократные побуждения к работе, без которых автор давно бы обленился.
- Издательству "БХВ-Петербург" — за издание моих книг.
- Всем своим читателям и почитателям — за прекрасные отзывы о моих книгах.
- Всем, кого я забыл здесь перечислить, — за все хорошее.
- Особая благодарность — программирующему народу!



ЧАСТЬ I

Общие приемы программирования



ГЛАВА 1

Полезные функции, методы и приемы программирования на JavaScript

В этой главе народ ответит на различные вопросы, связанные с самим языком JavaScript, а также предоставит несколько полезных функций и методов, которых в этом языке явно не хватает.

Работа с переменными и их значениями

Начнутся народные советы с решения проблем с переменными и хранящимися в них значениями.

Как проверить существование переменной?

Проблема

Как проверить, была ли уже объявлена какая-либо переменная?

Решение

Проще всего воспользоваться оператором `typeof`, возвращающим в строковом виде тип значения переменной. В случае еще не объявленной переменной он вернет строку "undefined".

Вот шаблон выражения, выполняющего проверку на существование переменной:

```
if (typeof(<Имя проверяемой переменной>) == "undefined")
  <Код, выполняемый, если переменная не объявлена>
else
  <Код, выполняемый, если переменная объявлена>
```

Пример

```
if (typeof(pi) == "undefined") pi = 3.14;
```

Как избежать конфликтов имен переменных?

Проблема

Имеется большой проект, содержащий множество глобальных переменных, причем различные части этого проекта написаны разными разработчиками. Имеется ли какой-либо способ избежать *конфликтов имен* переменных, т. е. исключить ситуации, когда один разработчик объявляет переменную с именем, уже использованным другим разработчиком в другой части того же сценария?

Решение

Решение здесь будет чисто организационное. Нужно просто продумать систему именования глобальных переменных и строго ей следовать. Система эта может быть, скажем, такой:

```
<Аббревиатура проекта>_<Аббревиатура части проекта>_<Тип данных>
[_<Подтип данных>]
```

Здесь:

- *Аббревиатура проекта* содержит наименование проекта (скажем, `emag` — электронный магазин);
- *Аббревиатура части проекта* указывает на часть проекта, разрабатываемую данным программистом (например, `checkData` — код, выполняющий проверку введенных покупателем данных о себе);
- *Тип данных* обозначает тип хранящегося в переменной значения (так, если переменная хранит сведения о покупателе, *Тип данных* можно назвать `custData`);
- необязательный *Подтип данных* можно использовать для указания на отдельную часть данных, относящихся к заданному ранее *Типу данных* (для имени покупателя это может быть `custName`).

> ***Народ советует***
>
> Эту систему можно расширить. Так, функции, реализующие методы объектов (о них мы поговорим далее в этой главе), могут иметь имена, начинающиеся на букву "m", а параметры функций — на букву "p". Также в имена функций, реализующих методы объектов, можно включить аббревиатуру, обозначающую объект, для которого они предназначены. Кстати, далее в своей книге автор так и делает.

Пример

```
var emag_checkData_custData_custName = getCustName();
```

Народ советует

Вообще, чем длиннее имя какой-либо переменной, тем меньше вероятность, что где-то в коде встретится другая переменная с таким же именем. Следует иметь это в виду.

Как создать константу?

Проблема

В коде сценария JavaScript часто используется одно и то же строковое или числовое значение. Можно ли вместо того, чтобы каждый раз указывать это значение, создать на его основе именованную *константу* и впоследствии использовать именно ее?

Решение

К сожалению, в отличие от других языков программирования, JavaScript не предоставляет никаких способов создать именованные константы. Но можно создать переменную, назвать ее каким-либо особым образом и присвоить ее нужное значение, создав, если можно так сказать, *псевдоконстанту*. Единственное "но": впоследствии не следует изменить значение этой переменной (константа она, в конце концов, или нет!).

Константы имеют одно неоспоримое преимущество. Если нам понадобится в дальнейшем изменить значение какой-нибудь константы, нам будет достаточно сделать это всего один раз — в выражении, объявляющем соответствующую переменную-псевдоконстанту. Если же мы не использовали константу, то будем вынуждены просмотреть все наши сценарии и исправить все использующие нужное нам значение выражения. А это зачастую очень трудоемко.

Пример

```
var EMAG_VERSION = "1.0";
. . .
document.write("Электронный магазин " + EMAG_VERSION);
. . .
document.write("Версия: " + EMAG_VERSION);
```

Этот сценарий сначала объявляет переменную-псевдоконстанту `EMAG_VERSION`, хранящую номер версии электронного магазина, а потом выводит этот номер на Web-страницу в двух местах.

Народ советует

Хорошим тоном программирования считается именование констант только большими буквами. Так мы, с одной стороны, сразу дадим своим коллегам понять, что это именно константа, а с другой — избежим возможных конфликтов имен.

Также хорошим тоном программирования считается вынесение всех глобальных констант, используемых в нескольких сценариях, в отдельный файл сценариев. Этот файл можно назвать, скажем, constants.js. Разумеется, этот файл придется потом подключить к Web-странице с помощью тега

```
<SCRIPT SRC="constants.js"></SCRIPT>
```

О файлах сценариев будет подробно рассказано в *главе 2*.

Работа с функциями

Здесь народ расскажет о различных полезных, но неочевидных приемах написания функций.

Как создать необязательные параметры функции?

Проблема

Есть функции, некоторые параметры которых очень часто принимают одни и те же значения. Нельзя ли сделать эти параметры *необязательными*, чтобы при вызове функции их можно было пропускать, и при этом они сами принимали бы нужные значения?

Решение

Если при вызове функции какой-либо из ее параметров был пропущен, в коде тела функции ему будет присвоено значение `undefined`. Так что последовательность действий ясна.

1. Проверяем, имеет ли этот параметр значение `undefined` (т. е. был ли он передан функции явно). Если да, переходим к шагу 2, если нет — к шагу 3.
2. Присваиваем параметру значение по умолчанию.
3. Используем значение этого параметра в вычислениях внутри тела функции.

Пример 1

Вот объявление функции, второй параметр которой — `par2` — является необязательным и имеющим значение по умолчанию 0.

```
function func1(par1, par2)
  {
    if (typeof(par2) == "undefined") par2 = 0;
    . . .
  }
```

Мы можем вызвать эту функцию как с указанием всех параметров:

```
func1(5, 2);
```

так и с указанием только первого — обязательного — параметра:

```
func1(10);
```

В последнем случае второй параметр получит в теле функции значение 0. Эффект будет тот же, если бы мы вызвали функцию с такими параметрами:

```
func1(10, 0);
```

Пример 2

Выражение проверки параметра на значение `undefined` в теле функции можно записать короче. Вот так:

```
function func1(par1, par2)
  {
    if (!par2) par2 = 0;
    . . .
  }
```

Значение `undefined`, будучи составной частью логического выражения, всегда преобразуется в значение `false`. В приведенном выше примере мы это и использовали.

> ***Внимание!***
>
> Необязательные параметры должны находиться в самом конце списка параметров функции (см. приведенные ранее примеры). Если же мы, например, сделаем необязательным первый параметр функции и вызовем ее таким образом:
>
> ```
> func1(, 2);
> ```
>
> пропустив его, это вызовет синтаксическую ошибку. Интерпретатор JavaScript не любит "пустых мест" в начале и середине списка параметров вызываемой функции.

Как передать функции произвольное число параметров?

Проблема

Нужно написать функцию, которая принимает произвольное число параметров. Как это сделать?

Решение 1

Внутри тела функции доступен массив `arguments`, элементы которого содержат все переданные функции параметры. Этот массив имеет свойство `length`, возвращающее количество элементов массива.

Пример 1

Вот объявление функции, принимающей произвольное количество строковых параметров и возвращающей строку, составленную из их значений:

```
function func2()
  {
    var s = "";
    var i = 0;
    var c = arguments.length;
    if (c > 0)
      for (i = 0; i < c; i++)
        s += arguments[i];
    return s;
  }
```

Выражение

```
var str = func2("Java", "Script");
```

поместит в переменную `str` строку "JavaScript".

Пример 2

Еще одна функция, принимающая произвольное количество строковых параметров и возвращающая строку, составленную из их значений. Но на этот раз первым, обязательным, параметром она принимает символ-разделитель, который будет помещаться между этими строками.

```
function func3(pDelimiter)
  {
    var s = "";
    var i = 0;
    var c = arguments.length;
    if (c > 1)
      for (i = 1; i < c; i++)
        {
          s += arguments[i];
          if (i != c - 1) s += pDelimiter;
        }
    return s;
  }
```

Выражение

```
var str = func3("+", "Java", "Script");
```

поместит в переменную `str` строку "Java+Script".

Решение 2

В конце концов, можно передать в функцию в качестве параметра массив с нужным количеством элементов.

Пример

Немного измененный пример функции `func3`:

```
function func3(pDelimiter, pars)
  {
    var s = "";
    var i = 0;
    var c = pars.length;
    if (c > 0)
      for (i = 0; i < c; i++)
        {
          s += pars[i];
          if (i != c - 1) s += pDelimiter;
        }
    return s;
  }
```

Вызываться эта функция будет так:

```
var str = func3(" ", new Array("Java", "Script"));
```

после чего в переменной `str` окажется строка "Java Script".

Возврат из функции нескольких значений

Проблема

Что делать, если какая-либо функция вычисляет не одно, а сразу несколько значений, и их все нужно вернуть? Стандартными средствами JavaScript (а именно оператором `return`) можно вернуть только одно значение.

Решение 1

1. Вычисляем в теле функции все нужные значения.
2. Создаем в теле функции массив.

3. Помещаем вычисленные значения в элементы этого массива.

4. Возвращаем полученный массив оператором `return`.

А уж извлечь значения элементов массива, возвращенного функцией, никакого труда не составит.

Примеры

Вот объявление функции, вычисляющей несколько значений и возвращающих их в массиве:

```
function someFunc()
  {
    // Здесь помещаются выражения, вычисляющие нужные значения.
    // Предположим, что всего их три, и они помещаются в переменные
    // value1, value2 и value3.
    var arr = new Array();
    arr[0] = value1;
    arr[1] = value2;
    arr[2] = value3;
    return arr;
  }
```

Использование этой функции:

```
var a = someFunc();
var v1 = a[0];
var v2 = a[1];
var v3 = a[2];
```

Вместо обычных массивов с числовыми индексами мы можем использовать *хэши* — массивы со строковыми индексами элементов. Например:

```
function someFunc()
  {
    // Здесь помещаются выражения, вычисляющие нужные значения
    // value1, value2 и value3.
    var arr = new Array();
    arr["value1"] = value1;
    arr["value2"] = value2;
    arr["value3"] = value3;
    return arr;
  }
var a = someFunc();
var v1 = a["value1"];
var v2 = a["value2"];
var v3 = a["value3"];
```

Пожалуй, так будет даже нагляднее, особенно если дать элементам массива вразумительные строковые индексы.

Народ советует

Если для возврата значений из функций используются хэши, логично оформить индексы их элементов в виде псевдоконстант. (Псевдоконстанты JavaScript были описаны ранее.) Например:

```
var JSPS_SFARR_1 = "value1";
var JSPS_SFARR_2 = "value2";
var JSPS_SFARR_3 = "value3";
```

А потом использовать для доступа к элементам хэша именно эти константы:

```
function someFunc()
  {
    // Здесь помещаются выражения, вычисляющие нужные значения
    // value1, value2 и value3.
    var arr = new Array();
    arr[JSPS_SFARR_1] = value1;
    arr[JSPS_SFARR_2] = value2;
    arr[JSPS_SFARR_3] = value3;
    return arr;
  }
var a = someFunc();
var v1 = a[JSPS_SFARR_1];
var v2 = a[JSPS_SFARR_2];
var v3 = a[JSPS_SFARR_3];
```

Решение 2

1. Вычисляем в теле функции все нужные значения.
2. Создаем в теле функции экземпляр объекта `Object`.
3. Помещаем вычисленные значения в свойства этого экземпляра.
4. Возвращаем полученный экземпляр оператором `return`.

Извлечь вычисленные функцией значения можно, обратившись к свойствам возвращенного экземпляра объекта `Object`.

Пример

Объявление функции, возвращающей несколько значений с помощью экземпляра объекта `Object`:

```
function someFunc()
  {
    // Здесь помещаются выражения, вычисляющие нужные значения
    // value1, value2 и value3.
```

```
    var obj = new Object();
    obj.value1 = value1;
    obj.value2 = value2;
    obj.value3 = value3;
    return obj;
  }
```

Использование ее:

```
var o = someFunc();
var v1 = o.value1;
var v2 = o.value2;
var v3 = o.value3;
```

Народ советует

Желающие могут даже создать особый объект, предназначенный специально для возврата из функции вычисленных ею значений. (Подробнее о создании объектов будет рассказано далее.)

```
function jspsSFReturnHolder(v1, v2, v3)
  {
    this.value1 = v1;
    this.value2 = v2;
    this.value3 = v3;
  }
function someFunc()
  {
    // Здесь помещаются выражения, вычисляющие нужные значения
    // value1, value2 и value3.
    var obj = new jspsSFReturnHolder(value1, value2, value3);
    return obj;
  }
```

Решение 3

Не столь изящное, как первые два, но используется, пожалуй, чаще всего.

1. Создаем массив или экземпляр объекта `Object`, который передается в функцию в качестве одного из параметров.
2. В теле функции заносим вычисленные ею значения в этот массив или экземпляр объекта.
3. После завершения работы функции извлекаем полученные значения из элементов массива или свойств экземпляра объекта.

При этом из функции вообще может не возвращаться никакое значение (имеется в виду возврат с помощью оператора `return`).

Пример

Объявление функции, возвращающей несколько значений в массиве, переданном ей в качестве параметра:

```
function someFunc(arr)
  {
    // Здесь помещаются выражения, вычисляющие нужные значения
    // value1, value2 и value3.
    arr[0] = value1;
    arr[1] = value2;
    arr[2] = value3;
  }
```

Ее использование:

```
var a = new Array();
someFunc(a);
var v1 = a[0];
var v2 = a[1];
var v3 = a[2];
```

Реализация статических переменных

Проблема

Локальные переменные, объявленные в теле функции, не сохраняют свои значения между вызовами этой функции. Но иногда бывает необходимо создать так называемую *статическую переменную* — локальную переменную, которая сохраняет свое значение между вызовами функции. Как это сделать?

Решение

Увы!.. JavaScript не предоставляет для этого стандартных средств. Нам придется объявить глобальную переменную и использовать ее в теле функции.

Пример

Приведенная далее функция сохраняет число своих вызовов в глобальной переменной `counter`.

```
var counter = 0;
function func4()
  {
    . . .
    counter++;
    . . .
  }
```

Работа с объектами

Здесь описаны некоторые приемы создания пользовательских и расширения встроенных объектов JavaScript.

Как создать пользовательский объект?

Проблема

Нужно создать пользовательский объект, имеющий определенные свойства и методы.

Решение

Объявление пользовательского объекта в JavaScript суть объявление особой функции, которая создает все его свойства и методы и заполняет их начальными значениями. Эта функция называется *конструктором*.

Формат объявления функции-конструктора ничем не отличается от формата объявления обычной функции:

```
function <Имя пользовательского объекта> ([<Список параметров>])
  {
    <Объявление свойств объекта>
    <Объявление методов объекта>
  }
```

Заметим, что имя функции-конструктора должно совпадать с именем объекта, который мы хотим создать.

Формат выражения, создающего новое свойство пользовательского объекта, таков:

```
this.<Имя свойства> = <Значение свойства>;
```

В данном случае ключевое слово `this` обозначает создаваемый объект.

> ***Внимание!***
>
> Ключевое слово `this` имеет смысл только в теле функции-конструктора.

Формат выражения, создающего метод, аналогичен:

```
this.<Имя метода> = <Функция, реализующая этот метод>;
```

Фактически здесь мы присваиваем функцию созданному свойству. Заметим при этом, что в данном случае имя функции указывается без списка параметров и без скобок. Разумеется, реализующая метод функция должна быть объявлена.

В функциях, реализующих методы создаваемого объекта, мы также можем использовать ключевое слово this. Собственно, даже не можем, а должны — ведь это единственный способ получить доступ к свойствам и методам этого объекта.

Созданный таким образом объект мы можем использовать в сценариях. Для создания на его основе экземпляра мы воспользуемся знакомым нам оператором new, а для удаления этого экземпляра — оператором delete.

Народ замечает

Описанный выше трюк с присвоением функции переменной работает и вне тела функции-конструктора. Так, мы можем присвоить любую функцию переменной:

```
function func5()
  {
    // Тело функции
  }
var varFunc = func5;
```

а потом вызвать ее, обратившись к этой переменной:

```
varFunc();
```

Повторим — при присвоении функции переменной имя присваиваемой функции указывается без списка параметров и без скобок, иначе интерпретатор JavaScript посчитает его за вызов этой функции.

Пример

Давайте создадим объект Point, обозначающий точку на экране. Этот объект будет содержать свойства x и y (координаты точки) и метод setCoords (помещение значений координат, переданных в качестве параметров, в соответствующие свойства).

Функция-конструктор этого объекта будет принимать два параметра — координаты точки. Эти координаты будут помещены в свойства x и y создаваемого экземпляра объекта.

```
// Объявление функции-конструктора объекта Point
function Point(ix, iy)
  {
    this.x = ix;
    this.y = iy;
    this.setCoords = fSetCoords;
  }
// Объявление функции, реализующей метод setCoords объекта Point
function fSetCoords(ix, iy)
```

```
  {
    this.x = ix;
    this.y = iy;
  }
```

Использование созданного объекта:

```
var p = new Point(50, 50);
p.setCoords(100, 200);
var px = p.x;
delete p;
```

Как создать пользовательский объект на основе уже имеющегося?

Проблема

Есть уже созданный пользовательский объект. Нужно создать на его основе другой объект, имеющий дополнительные свойства и методы.

Решение

Назовем уже имеющийся объект, на основе которого нужно создать новый, *объектом-предком*, создаваемый на его основе объект — *объектом-потомком*, а сам процесс порождения нового объекта на основе старого — *наследованием*. Свойства и методы, которые объект-потомок получит "в наследство" от объекта-предка, пусть называются *унаследованными*. Этими терминами пользуются все профессиональные программисты.

Первым шагом создания объекта-потомка будет написание его функции-конструктора. Ее формат будет таким:

```
function <Имя объекта-потомка> ([<Список параметров>])
  {
    this.base = <Имя объекта-предка>;
    this.base([<Список параметров функции-конструктора объекта-предка>]);
    <Определение новых свойств объекта-потомка>
    <Определение новых методов объекта-потомка>
    <Переопределение унаследованных свойств>
    <Переопределение унаследованных методов>
  }
<Имя объекта-потомка>.prototype = new <Объект-предок>;
```

Первое выражение тела функции-конструктора объекта-потомка присваивает функцию-конструктор объекта-предка свойству `base`. Это нужно, чтобы вызвать конструктор предка, дабы он создал все унаследованные свойства и методы и заполнил их значениями. Сам же вызов конструктора предка выполняется вторым выражением тела конструктора потомка.

Как создаются новые свойства и методы объекта-потомка, мы уже знаем. Кроме этого, мы можем переопределить значения унаследованных свойств и изменить унаследованные методы. Как изменить значение унаследованного свойства, понятно; для изменения же унаследованного метода достаточно присвоить ему другую функцию.

А теперь рассмотрим самое последнее выражение:

```
<Имя объекта-потомка>.prototype = new <Объект-предок>;
```

Оно требует особого внимания, т. к. дает понять интерпретатору JavaScript, что мы собираемся создать новый объект на основе существующего. Оно создает экземпляр объекта-предка и присваивает его особому свойству `prototype` объекта-потомка. Именно после этого объект-потомок унаследует все свойства и методы предка.

Пример

Давайте расширим созданный ранее объект `Point`. Во-первых, мы добавим к нему свойство `color`, хранящее цвет точки. Во-вторых, мы изменим метод `setCoords` — пусть при установке координат точки он присваивает свойству `color` значение 0 (черный цвет). Назовем новый объект `Point2`.

```
// Объявление функции-конструктора объекта Point2
function Point2(ix, iy, icolor)
  {
    this.base = Point;
    this.base(ix, iy);
    this.color = icolor;
    this.setCoords = fSetCoords2;
  }
// Выражение, делающее объект Point предком объекта Point2
Point2.prototype = new Point;
// Объявление функции, реализующей метод setCoords объекта Point2
function fSetCoords2(ix, iy)
  {
    this.x = ix;
    this.y = iy;
    this.color = 0;
  }
```

Теперь созданный объект можно использовать:

```
var p = new Point2(50, 50, 256);
p.setCoords(100, 200);
var px = p.x;
delete p;
```

Как вызвать метод объекта-предка из метода объекта-потомка?

Проблема

Есть пользовательский объект, созданный на основе другого объекта. Этот объект содержит метод, унаследованный от объекта-предка, но переопределенный в его собственной реализации. (Как мы уже знаем, это достигается присвоением другой функции соответствующему свойству объекта-потомка.) Можно ли из переопределенного метода потомка как-то вызвать изначальный метод предка?

Решение

Одно из ограничений языка JavaScript — невозможность стандартными средствами обратиться к изначальному методу объекта-предка из переопределенного метода потомка. Однако можно использовать несложный трюк, описанный далее.

1. В определении объекта-потомка создаем еще одно свойство и называем его, скажем, по такой схеме: *<Имя объекта-предка>_<Имя переопределяемого метода>*.
2. Присваиваем этому свойству значение свойства, соответствующего переопределяемому методу.
3. Присваиваем свойству, соответствующему переопределяемому методу, функцию, содержащую новую реализацию этого метода.
4. Если нам понадобится обратиться к изначальному методу предка, мы просто вызываем его, пользуясь созданным ранее свойством *<Имя объекта-предка>_<Имя переопределяемого метода>*.

Пример

Давайте перепишем определение объекта `Point2` так, чтобы функция `fSetCoords2`, заносящая в свойства `x` и `y` значения координат, использовала для этого "старую" функцию `fSetCoords`.

```
// Объявление функции-конструктора объекта Point2
function Point2(ix, iy, icolor)
  {
    this.base = Point;
    this.base(ix, iy);
    this.color = icolor;
    this.Point_setCoords = this.setCoords;
    this.setCoords = fSetCoords2;
  }
```

```
// Выражение, делающее объект Point предком объекта Point2
Point2.prototype = new Point;
// Объявление функции, реализующей метод setCoords объекта Point2
function fSetCoords2(ix, iy)
  {
    this.Point_setCoords(ix, iy);
    this.color = 0;
  }
```

Народ советует

Несмотря, скажем так, на особенности реализации в JavaScript, использование изначальных методов объектов-предков может сильно упростить код методов объектов-потомков. Тем более что полностью изменять реализацию переопределяемых методов приходится очень редко — чаще их нужно только слегка подправить.

Как добавить новые свойства и методы в уже существующий объект?

Проблема

Я написал отличную функцию по работе со строками. Нельзя ли добавить ее к уже существующему объекту `String` в качестве метода?

Решение

Ничего нет проще! Синтаксис создания нового свойства или метода для уже существующего объекта таков:

```
<Объект>.prototype.<Имя свойства> = <Значение свойства>;
<Объект>.prototype.<Имя метода> = <Функция, реализующая этот метод>;
```

Народ предупреждает!

Описанным выше способом можно добавить новые свойства и методы только во встроенные объекты JavaScript (`Number`, `String`, `Object` и пр.). Объекты Web-обозревателя (`document`, `window`, `navigator` и пр.) таким образом расширить нельзя.

Пример

Далее в этой книге будут приведены листинги методов `trimLeft` и `trimRight`, удаляющие из строк начальные и конечные пробелы соответственно. Для добавления этих методов к объекту `String` были использованы следующие выражения:

```
String.prototype.trimLeft = mjspsTrimLeft;
String.prototype.trimRight = mjspsTrimRight;
```

Здесь mjspsTrimLeft и mjspsTrimRight — функции, реализующие эти методы (они будут описаны далее).

Как в теле метода, добавленного к объекту *String*, *Number* или *Boolean*, получить доступ к значению этого объекта?

Проблема

Я пишу метод, который собираюсь добавить к стандартному объекту JavaScript (String, Number или Boolean). Как мне в этом методе получить текущее значение этого объекта (строку, число или логическую величину)?

Решение

Общий формат выражения для получения текущего значения стандартного объекта JavaScript в теле метода этого объекта таков:

```
<Переменная> = new <Имя стандартного объекта>(this);
```

Пример

Вот фрагмент объявления функции, которая впоследствии станет методом объекта Number:

```
function mSomeMethod()
  {
    var objValue = new Number(this);
    . . .
  }
```

В результате вычисления приведенного выше единственного выражения в переменной objValue окажется числовое значение, содержащееся в экземпляре объекта Number, т. е. в числовой переменной.

Манипуляции строковыми значениями

Здесь собраны предложенные народом функции и методы объекта String, которые могут пригодиться при работе со строками.

> ***Хорошая идея!***
>
> Поместите объявления функций и методов, работающих со строками, в файл сценариев. Этот файл можно назвать, например, stringutils.js. Впоследствии, чтобы использовать какую-либо функцию или метод, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:
>
> ```
> <SCRIPT SRC="stringutils.js"></SCRIPT>
> ```

Преобразование нестрокового значения в строковый вид

Проблема

Как преобразовать значение, не являющееся строкой (например, число), в строковый вид?

Решение

Использование метода `toString()`, поддерживаемого всеми встроенными объектами JavaScript. Этот метод не принимает параметров и возвращает строковое представление значения, хранящегося в переменной.

Примеры

```
var i = 10;
var f = 2.53
var b = true;
var si = i.toString();
var sf = f.toString();
var sb = b.toString();
```

После выполнения этих выражений в переменных `si`, `sf` и `sb` окажутся, соответственно, строки "10", "2.53" и "true".

Удаление пробелов в начале и конце строки

Проблема

Очень часто приходится удалять пробелы в начале и конце строки. JavaScript не имеет для этого ни встроенной функции, ни метода объекта `String`.

Решение 1

Использование функций `jspsTrimLeft` (листинг 1.1) и `jspsTrimRight` (листинг 1.2), форматы вызова которых таковы:

```
jspsTrimLeft(<Строка>);
jspsTrimRight(<Строка>);
```

Обе функции возвращают обработанную строку.

Листинг 1.1. Функция `jspsTrimLeft`, возвращающая строку без начальных пробелов

```
function jspsTrimLeft(workString)
  {
    var i = 0;
```

```
    var c = workString.length;
    if (c > 0)
      {
        while ((i < c) && (workString.charAt(i) == " ")) i++;
        if (i == c)
          workString = ""
        else
          workString = workString.substring(i, c);
      }
    return workString;
  }
```

Листинг 1.2. Функция `jspsTrimRight`, возвращающая строку без конечных пробелов

```
function jspsTrimRight(workString)
  {
    var c = workString.length;
    var i = c - 1;
    if (c > 0)
      {
        while ((i >= 0) && (workString.charAt(i) == " ")) i--;
        if (i == -1)
          workString = ""
        else
          workString = workString.substring(0, i + 1);
      }
    return workString;
  }
```

Также нам будет полезна функция `jspsTrim` (листинг 1.3), удаляющая из строки и начальные, и конечные пробелы. Формат ее вызова схож с форматом вызова описанных выше функций.

Листинг 1.3. Функция `jspsTrim`, возвращающая строку без начальных и конечных пробелов

```
function jspsTrim(workString)
  {
    return jspsTrimRight(jspsTrimLeft(workString));
  }
```

Внимание!

Листинг 1.3, содержащий объявление функции `jspsTrim`, использует также функции `jspsTrimLeft` и `jspsTrimRight`, чьи объявления приведены в листингах 1.1 и 1.2 соответственно.

Пример

```
var userName = "    sysadmin  ";
var userName = jspsTrim(userName);
```

В результате выполнения этого сценария в переменной `userName` окажется строка "sysadmin".

Решение 2

Использование методов `trimLeft` (листинг 1.4), `trimRight` (листинг 1.5) и `trim` (листинг 1.6) объекта `String`, форматы вызова которых:

```
<Строка>.trimLeft;
<Строка>.trimRight;
<Строка>.trim;
```

Все эти методы возвращают обработанную строку.

Листинг 1.4. Метод `trimLeft` объекта `String`, возвращающий строку без начальных пробелов

```
function mjspsTrimLeft()
  {
    var workString = new String(this);
    var i = 0;
    var c = workString.length;
    if (c > 0)
      {
        while ((i < c) && (workString.charAt(i) == " ")) i++;
        if (i == c)
          workString = ""
        else
          workString = workString.substring(i, c);
      }
    return workString;
  }

String.prototype.trimLeft = mjspsTrimLeft;
```

Листинг 1.5. Метод `trimRight` объекта `String`, возвращающий строку без конечных пробелов

```
function mjspsTrimRight()
  {
    var workString = new String(this);
```

```
    var c = workString.length;
    var i = c - 1;
    if (c > 0)
      {
        while ((i >= 0) && (workString.charAt(i) == " ")) i--;
        if (i == -1)
          workString = ""
        else
          workString = workString.substring(0, i + 1);
      }
    return workString;
  }

String.prototype.trimRight = mjspsTrimRight;
```

Листинг 1.6. Метод `trim` объекта `String`, возвращающий строку без начальных и конечных пробелов

```
function mjspsTrim()
  {
    var workString = new String(this);
    return jspsTrimRight(jspsTrimLeft(workString));
  }

String.prototype.trim = mjspsTrim;
```

Внимание!

Листинг 1.6, содержащий объявление метода `trim` объекта `String`, использует также методы `trimLeft` и `trimRight` того же объекта, чьи объявления приведены в листингах 1.4 и 1.5 соответственно.

Пример

```
var userName = "    sysadmin  ";
var userName = userName.trim();
```

В результате выполнения этого сценария в переменной `userName` окажется строка "sysadmin".

Народ советует

Всегда по возможности следует дублировать написанные функции аналогичными методами. Дело в том, что одни программисты ("старой" школы) предпочитают работать с функциями, другие (поклонники объектно-ориентированного стиля) — с методами, и таким образом мы дадим им возможность выбора.

Подсчет всех вхождений подстроки в строку

Проблема

Очень часто бывает нужно подсчитать, сколько раз в заданную строку входит какая-либо подстрока. Как это сделать?

Решение 1

Использование функции `jspsAllMatches` (листинг 1.7), формат вызова которой выглядит так:

```
jspsAllMatches(<Строка>, <Искомая подстрока>);
```

Собственно, объяснений здесь особых не требуется. Первым параметром передается *Строка*, в которой выполняется поиск всех вхождений подстроки. Вторым же параметром передается сама *Искомая подстрока*. Возвращает функция число вхождений *Искомой подстроки* в *Строку*.

Листинг 1.7. Функция `jspsAllMatches`, возвращающая число вхождений подстроки в строке

```
function jspsAllMatches(workString, searchSubstring)
  {
    var nSS = 0;
    var nWSLen = workString.length;
    var nSSLen = searchSubstring.length;
    var nP = workString.indexOf(searchSubstring);
    while (nP != -1)
      {
        nSS++;
        nP += nSSLen;
        nP = workString.indexOf(searchSubstring, nP);
      }
    return nSS;
  }
```

Пример

```
var s = "JavaScript";
var n = jspsAllMatches(s, "a");
```

Приведенный в листинге 1.7 сценарий поместит в переменную `n` число 2 — это количество букв "a" в слове "JavaScript".

Решение 2

Использование метода `allMatches` объекта `String` (листинг 1.8), формат вызова которого:

```
<Строка>.allMatches(<Искомая подстрока>);
```

Листинг 1.8. Метод `allMatches` объекта `String`, возвращающий число вхождений подстроки в строке

```
function mjspsAllMatches(searchSubstring)
  {
    var workString = new String(this);
    var nSS = 0;
    var nWSLen = workString.length;
    var nSSLen = searchSubstring.length;
    var nP = workString.indexOf(searchSubstring);
    while (nP != -1)
      {
        nSS++;
        nP += nSSLen;
        nP = workString.indexOf(searchSubstring, nP);
      }
    return nSS;
  }

String.prototype.allMatches = mjspsAllMatches;
```

Пример

```
var s = "JavaScript";
var n = s.allMatches("a");
```

Замены подстроки другой подстрокой

Проблема

Очень часто приходится выполнять замену подстроки в строке другой подстрокой. Язык JavaScript не содержит для этого никаких встроенных средств, что сильно осложняет задачу программистов.

Решение 1

Использование функции `jspsStringReplace` (листинг 1.9), формат вызова которой выглядит так:

```
jspsStringReplace(<Строка>, <Искомая подстрока>, <Заменяющая подстрока>
↳[, <Номер вхождения искомой подстроки>]);
```

Эта функция принимает четыре параметра. Первым параметром передается собственно *Строка*, которую нужно обработать с помощью этой функции. Второй и третий параметры в объяснении не нуждаются. Что касается четвертого, необязательного, параметра, то он задает номер вхождения *Искомой подстроки* в *Строку*, которое нужно заменить (подстроки нумеруются, начиная с единицы). Если этот параметр не задан, выполняется замена первой найденной *Искомой подстроки*.

Функция возвращает обработанную строку, в которой найденное вхождение *Искомой подстроки* заменено. Если *Искомая подстрока* в *Строке* не встретилась, возвращается неизмененная *Строка*.

Листинг 1.9. Функция `jspsStringReplace`, заменяющая заданную подстроку другой подстрокой

```
function jspsStringReplace(workString, searchSubstring, replaceSubstring,
↳substringNumber)
  {
    var s1, s2, nSS = 1;
    if (!substringNumber) substringNumber = 1;
    var nWSLen = workString.length;
    var nSSLen = searchSubstring.length;
    var nP = workString.indexOf(searchSubstring);
    while (nP != -1)
      {
        if (nSS == substringNumber)
          {
            s1 = workString.substring(0, nP);
            s2 = workString.substring(nP + nSSLen, nWSLen);
            workString = s1 + replaceSubstring + s2;
            break;
          }
        nP = workString.indexOf(searchSubstring, nP + 1);
        nSS++;
      }
    return workString;
  }
```

Примеры

```
var s1 = "Здравствуйте, <имя посетителя>!";
var s2 = jspsStringReplace(s1, "<имя посетителя>", "Вася Пупкин");
document.write(s2);
```

Этот сценарий выведет на Web-страницу строку "Здравствуйте, Вася Пупкин".

```
document.write(jspsStringReplace("Текущий день - день", "день",
↳"понедельник", 2));
```

Это выражение выведет на Web-страницу строку "Текущий день - понедельник". Обратите внимание — функция выполнит замену второй подстроки "день" в заданной строке, т. к. последним, необязательным, параметром, задающим номер заменяемой подстроки, было передано число 2.

> ***Народ советует***
>
> Предусматривайте в своих функциях и методах необязательные параметры и подходящие значения по умолчанию для них. В конце концов, какие-либо сложные задачи выполняются программистом весьма редко — большей же частью они очень просты. Так, с помощью функции `jspsStringReplace` выполняется, в основном, замена единственной подстроки в строке, поэтому последний параметр, задающий номер заменяемой подстроки, сделан необязательным, принимающим значение по умолчанию 1.

Решение 2

Использование метода `stringReplace` (листинг 1.10) объекта `String`. Вот формат его вызова:

```
<Строка>.stringReplace(<Искомая подстрока>, <Заменяющая
↳подстрока>[, <Номер вхождения искомой подстроки>]);
```

Листинг 1.10. Метод `stringReplace` объекта `String`, заменяющий заданную подстроку другой подстрокой

```
function mjspsStringReplace(searchSubstring, replaceSubstring,
↳substringNumber)
  {
    var s1, s2, nSS = 1;
    var workString = new String(this);
    if (!substringNumber) substringNumber = 1;
    var nWSLen = workString.length;
    var nSSLen = searchSubstring.length;
    var nP = workString.indexOf(searchSubstring);
    while (nP != -1)
      {
        if (nSS == substringNumber)
          {
            s1 = workString.substring(0, nP);
            s2 = workString.substring(nP + nSSLen, nWSLen);
```

```
            workString = s1 + replaceSubstring + s2;
            break;
          }
        nP = workString.indexOf(searchSubstring, nP + 1);
        nSS++;
      }
    return workString;
  }

String.prototype.stringReplace = mjspsStringReplace;
```

Пример

```
document.write("Текущий день - день".stringReplace("день",
↳"понедельник", 2));
```

Замена всех подстрок в строке

Проблема

Функция `jspsStringReplace` и метод `stringReplace` объекта `String` позволяют заменить только одно-единственное вхождение заданной подстроки. А если таких подстрок в строке несколько, и заменить нужно их все "хором"?

Решение 1

Использование функции `jspsStringReplaceAll` (листинг 1.11), формат вызова которой приведен далее:

```
jspsStringReplaceAll(<Строка>, <Искомая подстрока>, <Заменяющая
↳подстрока>);
```

Назначение ее параметров понятно без объяснений. Функция возвращает обработанную строку, в которой все найденные вхождения *Искомой подстроки* заменены на *Заменяющую подстроку*. Опять же, если *Искомая подстрока* в *Строке* не встретилась, возвращается неизмененная *Строка*.

Листинг 1.11. Функция `jspsStringReplaceAll`, заменяющая все заданные подстроки в строке. Немного измененный вариант функции, приведенной в [1]

```
function jspsStringReplaceAll(workString, searchSubstring,
↳replaceSubstring)
  {
    var s1, s2;
    var nWSLen = workString.length;
    var nSSLen = searchSubstring.length;
    var nP = workString.indexOf(searchSubstring);
```

```
    while (nP != -1)
      {
        s1 = workString.substring(0, nP);
        s2 = workString.substring(nP + nSSLen, nWSLen);
        workString = s1 + replaceSubstring + s2;
        nP = workString.indexOf(searchSubstring);
      }
    return workString;
  }
```

Пример

```
var s = document.all["par"].innerHTML;
s = jspsStringReplaceAll(s, "<B>", "<I>");
s = jspsStringReplaceAll(s, "</B>", "</I>");
document.all["par"].innerHTML = s;
```

Этот сценарий извлечет HTML-код, содержащийся внутри тега, имеющего значение атрибута `ID`, равное "par", заменит в этом коде открывающие и закрывающие теги `<B>` на теги `<I>` и снова поместит в этот тег.

> ***Внимание!***
>
> Приведенный пример работает только в Microsoft Internet Explorer и Opera. Firefox не поддерживает свойство `innerHTML`.

Решение 2

Использование метода `stringReplaceAll` объекта `String` (листинг 1.12). Он имеет такой формат вызова:

```
<Строка>.stringReplaceAll(<Искомая подстрока>, <Заменяющая подстрока>);
```

Листинг 1.12. Метод `stringReplaceAll` объекта `String`, заменяющий все заданные подстроки в строке

```
function mjspsStringReplaceAll(searchSubstring, replaceSubstring)
  {
    var s1, s2;
    var workString = new String(this);
    var nWSLen = workString.length;
    var nSSLen = searchSubstring.length;
    var nP = workString.indexOf(searchSubstring);
    while (nP != -1)
      {
        s1 = workString.substring(0, nP);
        s2 = workString.substring(nP + nSSLen, nWSLen);
```

```
        workString = s1 + replaceSubstring + s2;
        nP = workString.indexOf(searchSubstring);
      }
    return workString;
  }

String.prototype.stringReplaceAll = mjspsStringReplaceAll;
```

Пример

```
var s = document.all["par"].innerHTML;
s = s.stringReplaceAll("<B>", "<I>");
s = s.stringReplaceAll("</B>", "</I>");
document.all["par"].innerHTML = s;
```

Внимание!

Приведенный пример работает только в Microsoft Internet Explorer и Opera. Firefox не поддерживает свойство `innerHTML`.

Форматированный вывод значений в строковом виде

Проблема

Имеется некоторая строка-шаблон и набор числовых и строковых значений. Можно ли расставить эти значения в нужных местах строки-шаблона?

Решение

Использовать функцию `jspsFormatStr` (листинг 1.13), формат вызова которой выглядит так:

```
jspsFormatStr(<Строка-шаблон>, <Массив значений>);
```

С первым параметром все понятно — в нем передается *Строка-шаблон*. Вторым же параметром передается *Массив значений*. Его элементы представляют собой значения, которые надо вставить в нужные места *Строки-шаблона*. Каждое из этих мест задается особым *литералом* (специальным символом), который включается в текст *Строки-шаблона*. Литерал этот имеет такой вид: `%s%`.

Функция возвращает готовую строку, в нужные места которой вставлены значения, переданные в виде элементов *Массива значений*. Если элементов в *Массиве значений* меньше, чем литералов, оставшиеся литералы будут удалены. Если же *Массив значений* содержит элементов больше, чем литералов в *Строке-шаблоне*, лишние элементы массива будут проигнорированы.

Листинг 1.13. Функция `jspsFormatStr`, выполняющая форматированный вывод значений в строковом виде

```
function jspsFormatStr(templateString, valuesArray)
  {
    var i = 0;
    var c = valuesArray.length;
    if (c > 0)
      for (i = 0; i < c; i++)
        templateString = jspsStringReplace(templateString, "%s%",
        ↳valuesArray[i].toString());
    templateString = jspsStringReplaceAll(templateString, "%s%", "");
    return templateString;
  }
```

Внимание!

Листинг 1.13, содержащий объявление функции `jspsAllMatches`, использует также функции `jspsStringReplace` и `jspsStringReplaceAll`, чьи объявления приведены в листингах 1.9 и 1.11 соответственно.

Пример

```
var s = "Java%s% %s% %s%";
var ss1 = "Script";
var ss2 = "v1.1";
var ss3 = "language"
document.write(jspsFormatStr(s, new Array(ss1, ss2, ss3)));
```

Приведенный сценарий поместит на Web-страницу текст "JavaScript v1.1 language".

Манипуляции числовыми значениями

Здесь собраны предложенные народом функции и методы объекта `Number`, которые могут пригодиться при работе с числовыми значениями.

Хорошая идея!

Поместите объявления функций и методов, работающих с числовыми значениями, в файл сценариев. Этот файл можно назвать, например, numberutils.js. Впоследствии, чтобы использовать какую-либо функцию или метод, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="numberutils.js"></SCRIPT>
```

Преобразование строки в числовое значение

Проблема

Как преобразовать строку, содержащую целое или дробное число, в числовое значение?

Решение

Использовать функции `parseInt` и `parseFloat`. Первая функция преобразует строку в целое число, вторая — в число с плавающей точкой.

Формат вызова функции `parseInt`:

```
parseInt(<Строка>[, <Основание>]);
```

Первым параметром в эту функцию передается сама строка, содержащая число. Вторым же параметром передается основание системы счисления; если он пропущен, интерпретатор JavaScript использует основание 10 (т. е. десятичную систему).

Формат вызова функции `parseFloat` проще:

```
parseFloat(<Строка>);
```

Обе функции возвращают число.

> ***Народ предупреждает!***
>
> Если число, переданное в функцию `parseInt` или `parseFloat`, невозможно преобразовать в число, возвращается особое значение `NaN` (Not a Number, не число).
>
> JavaScript предоставляет возможность проверить, можно ли преобразовать строку в число. Для этого используется функция `isNaN`. В качестве единственного параметра в нее передается строка. Если эта строка может быть преобразована в число, возвращается `true`, в противном случае — `false`.

Примеры

```
var n1 = parseInt("45");
var n2 = parseInt("3.14");
var n3 = parseInt("0xFF", 16);
var n4 = parseInt("JavaScript");
```

После выполнения этого сценария в переменной `n1` окажется число 45, в переменной `n2` — число 3 (поскольку функция `parseInt` "занимается" исключительно целыми числами), в переменной `n3` — число 255 (десятичное представление шестнадцатеричного числа FF), а в переменной `n4` — значение `NaN` (поскольку строка "JavaScript" не может быть никаким образом преобразована в число).

```
var f1 = parseFloat("3.14");
var f2 = parseFloat("4.5ggg");
```

А этот сценарий поместит в переменную `f1` число 3,14, а в переменную `f2` — число 4,5 (интерпретатор JavaScript сочтет первые три символа переданной в функцию `parseFloat` строки числом, а остальные символы отбросит).

Округление числа до произвольного знака после запятой

Проблема

JavaScript не предоставляет никаких средств для округления дробных чисел до произвольного знака после запятой. Как быть?

Решение

Использовать функцию `jspsRound` (листинг 1.14) или метод `round` объекта `Number` (листинг 1.15). Форматы их вызова таковы:

```
jspsRound(<Число>[, <Количество знаков после запятой>]);
round([<Количество знаков после запятой>]);
```

Если *Количество знаков после запятой* не указано, выполняется округление до целого числа. И функция, и метод возвращают округленное число.

Листинг 1.14. Функция `jspsRound`, возвращающая число, округленное до заданного количества знаков после запятой

```
function jspsRound(workNumber, precision)
  {
    if (!precision) precision = 0;
    var nM = Math.pow(10, precision);
    return Math.round(workNumber * nM) / nM;
  }
```

Листинг 1.15. Метод `round` объекта `Number`, возвращающий число, округленное до заданного количества знаков после запятой

```
function mjspsRound(precision)
  {
    if (!precision) precision = 0;
    var workNumber = new Number(this);
    var nM = Math.pow(10, precision);
    return Math.round(workNumber * nM) / nM;
  }

Number.prototype.round = mjspsRound;
```

Примеры

```
var f = 3.1415926;
var f1 = jspsRound(n, 4);
var f2 = n.round();
```

Приведенный сценарий поместит в переменную `f1` число 3,1416, а в переменную `f2` — число 3.

Преобразование числа в шестнадцатеричную и восьмеричную системы счисления

Проблема

Мне нужно вывести на Web-страницу число в шестнадцатеричной (восьмеричной) системе счисления. Как это сделать?

Решение

Использовать функцию `jspsToHex` (листинг 1.16). Формат ее вызова очень прост:

```
jspsToHex(<Число>);
```

Функция возвращает шестнадцатеричное число в строковом виде.

Вот заодно и функция `jspsToOctal` (листинг 1.17), преобразующая число в восьмеричную систему счисления:

```
jspsToOctal(<Число>);
```

Преобразованное число также возвращается в строковом виде.

Листинг 1.16. Функция `jspsToHex`, преобразующая число в шестнадцатеричную систему счисления

```
function jspsToHex(workNumber)
  {
    var cHex = new Array("0", "1", "2", "3", "4", "5", "6", "7", "8",
    ↳"9", "A", "B", "C", "D", "E", "F");
    var b = 16;
    var n1, n2, i;
    var s = "";
    while (workNumber > b) b *= 16;
    while (b > 1)
      {
        n1 = workNumber % b;
        n2 = (workNumber - n1) / b;
        s += cHex[n2];
        workNumber -= n2 * b;
```

```
        b /= 16;
      }
    s += cHex[workNumber];
    if (s.charAt(0) == "0") s = s.substring(1, s.length);
    return "0x" + s;
  }
```

Листинг 1.17. Функция `jspsToOctal`, преобразующая число в восьмеричную систему счисления

```
function jspsToOctal(workNumber)
  {
    var b = 8;
    var n1, n2, i;
    var s = "";
    while (workNumber > b) b *= 8;
    while (b > 1)
      {
        n1 = workNumber % b;
        n2 = (workNumber - n1) / b;
        s += n2.toString();
        workNumber -= n2 * b;
        b /= 8;
      }
    s += workNumber.toString();
    if (s.charAt(0) != "0") s = "0" + s;
    return s;
  }
```

Примеры

```
var n = 255;
var nh = jspsToHex(n);
var no = jspsToOctal(n);
```

В результате выполнения этого сценария в переменной `nh` окажется строка "0xFF" (шестнадцатеричное представление числа 255), а в переменной `no` — строка "0377" (восьмеричное представление того же числа).

Получение псевдослучайного числа в заданном диапазоне

Проблема

Требуется получить псевдослучайное число в заданном диапазоне чисел. Метод `random` объекта `Math` выдает только числа в диапазоне от 0 до 1.

Решение

Использовать функцию `jspsGetRandom` (листинг 1.18), формат вызова которой приведен ниже:

```
jspsGetRandom(<Минимальное значение диапазона>,
<Максимальное значение диапазона>);
```

Функция возвращает псевдослучайное число, находящееся в диапазоне между *Минимальным значением* и *Максимальным значением*.

Листинг 1.18. Функция `jspsGetRandom`, возвращающая псевдослучайное число, находящееся в диапазоне между заданными значениями

```
function jspsGetRandom(minValue, maxValue)
  {
    return minValue + (maxValue - minValue) * Math.random();
  }
```

Пример

```
for (var i = 0; i < 10; i++)
  document.write(jspsGetRandom(10, 20) + "<BR>");
```

Приведенный сценарий выведет на Web-страницу десять псевдослучайных чисел, расположенных в диапазоне от 10 до 20.

Преобразование величины угла из градусов в радианы и наоборот

Проблема

Методы вычисления тригонометрических функций объекта `Math` принимают параметры только в радианах. Каким образом можно преобразовать привычные мне величины из градусов в радианы и наоборот?

Решение

Использовать функции `jspsToRad` (листинг 1.19) и `jspsToDeg` (листинг 1.20). Первая функция выполняет преобразование градусов в радианы, вторая — из радианов в градусы. Форматы вызова этих функций таковы:

```
jspsToRad(<Величина в градусах>);
jspsToDeg(<Величина в радианах>);
```

Первая функция возвращает величину в радианах, вторая — в градусах.

Листинг 1.19. Функция `jspsToRad`, выполняющая преобразование из градусов в радианы

```
function jspsToRad(degValue)
  {
    return degValue * Math.PI / 180;
  }
```

Листинг 1.20. Функция `jspsToDeg`, выполняющая преобразование из радианов в градусы

```
function jspsToDeg(radValue)
  {
    return radValue * 180 / Math.PI;
  }
```

Примеры

```
var n1 = 180;
var n2 = 45;
var f1 = jspsToRad(n1);
var n2 = jspsToRad(n2);
```

Этот сценарий поместит в переменную `f1` число 3,141592653589793, а в переменную `f2` — число 0,7853981633974483. Это и будут значения соответствующих углов в радианах.

```
var f1 = Math.PI;
var f2 = Math.PI / 4;
var n1 = jspsToDeg(f1);
var n2 = jspsToDeg(f2);
```

А этот сценарий поместит в переменную `n1` число 180, а в переменную `n2` — число 45.

Манипуляции значениями даты и времени

Здесь собраны все функции и методы объекта `Date`, которые обязательно пригодятся при работе со значениями даты и времени.

Хорошая идея!

Создайте файл сценариев datetimeutils.js и поместите в него объявления функций и методов, манипулирующих значениями даты и времени. Впоследствии, чтобы использовать какую-либо функцию или метод, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="datetimeutils.js"></SCRIPT>
```

Форматированный вывод значений даты и времени

Проблема

Очень часто бывает нужно вывести на Web-страницу дату в привычном нам формате "число.месяц.год" или время в виде "часы:минуты:секунды". Стандартные же методы `toString` и `toLocaleString` объекта `Date` возвращают длинную и не очень красивую строку вида "Fri Apr 28 11:29:04 UTC+0400 2006" или "28 апреля 2006 г. 11:29:44" соответственно.

Решение

Использование универсального метода `toFormattedString` (листинг 1.21) объекта `Date`, реализующего форматированный вывод значений даты и времени в строковом формате. Вот формат его вызова:

```
<Дата>.toFormattedString([<Формат даты>]);
```

Он принимает единственный необязательный параметр *Формат даты*, задающий... — правильно! — формат даты. Он представляет собой строку, содержащую литералы, задающие число, месяц, год, часы, минуты и секунды. Все доступные для использования литералы:

- "DA" — число;
- "YD" — номер дня недели;
- "YN" — название дня недели;
- "MO" — номер месяца;
- "MN" — название месяца;
- "YE" — год;
- "HO" — часы;
- "MI" — минуты;
- "SE" — секунды;
- "AM" — если присутствует, время выводится в 12-часовом формате, если отсутствует — в 24-часовом ("военном").

Если же формат не задан, будет использован формат по умолчанию "DA.MO.YE", т. е. "число.месяц.год".

Метод возвращает строку, содержащую отформатированное значение даты.

> ***Внимание!***
>
> Листинг 1.21, содержащий исходный код метода `toFormattedString` объекта `Date`, использует функцию `jspsStringReplaceAll`, чей исходный код содержится в листинге 1.11.

Листинг 1.21. Метод `toFormattedString` объекта `Date`, возвращающий отформатированное значение даты

```
var JSPS_DAY_NAMES = new Array();
JSPS_DAY_NAMES[1] = "понедельник";
JSPS_DAY_NAMES[2] = "вторник";
JSPS_DAY_NAMES[3] = "среда";
JSPS_DAY_NAMES[4] = "четверг";
JSPS_DAY_NAMES[5] = "пятница";
JSPS_DAY_NAMES[6] = "суббота";
JSPS_DAY_NAMES[7] = "воскресенье";

var JSPS_MONTH_NAMES_2 = new Array();
JSPS_MONTH_NAMES_2[0] = "января";
JSPS_MONTH_NAMES_2[1] = "февраля";
JSPS_MONTH_NAMES_2[2] = "марта";
JSPS_MONTH_NAMES_2[3] = "апреля";
JSPS_MONTH_NAMES_2[4] = "мая";
JSPS_MONTH_NAMES_2[5] = "июня";
JSPS_MONTH_NAMES_2[6] = "июля";
JSPS_MONTH_NAMES_2[7] = "августа";
JSPS_MONTH_NAMES_2[8] = "сентября";
JSPS_MONTH_NAMES_2[9] = "октября";
JSPS_MONTH_NAMES_2[10] = "ноября";
JSPS_MONTH_NAMES_2[11] = "декабря";

function mjspsToFormattedString(fmString)
  {

    function normalizeN(n)
      {
        var s = n.toString();
        if (s.length == 1) s = "0" + s;
        return s;
      }

    var n;
    if (!fmString) fmString = "DA.MO.YE";
    fmString = jspsStringReplaceAll(fmString, "DA",
    ↳this.getDate().toString());
    n = this.getDay();
    if (n == 0) n = 7;
    fmString = jspsStringReplaceAll(fmString, "YD", n.toString());
```

```
    fmString = jspsStringReplaceAll(fmString, "YN", JSPS_DAY_NAMES[n]);
    fmString = jspsStringReplaceAll(fmString, "MO",
    ↳normalizeN(this.getMonth() + 1));
    fmString = jspsStringReplaceAll(fmString, "MN",
    ↳JSPS_MONTH_NAMES_2[this.getMonth()]);
    fmString = jspsStringReplaceAll(fmString, "YE",
    ↳this.getYear().toString());
    n = this.getHours();
    if (fmString.indexOf("AM") != -1)
      if (n > 12)
        {
          n = n - 12;
          fmString = jspsStringReplaceAll(fmString, "AM", "PM");
        }
    fmString = jspsStringReplaceAll(fmString, "HO", n.toString());
    fmString = jspsStringReplaceAll(fmString, "MI",
    ↳normalizeN(this.getMinutes()));
    fmString = jspsStringReplaceAll(fmString, "SE",
    ↳normalizeN(this.getSeconds()));
    return fmString;
  }

Date.prototype.toFormattedString = mjspsToFormattedString;
```

Пример

```
var d = new Date();
document.write(d.toFormattedString("DA.MO (MN).YE, день - YD (YN)
↳HO:MI:SE"));
```

Приведенный сценарий поместит на Web-страницу строку вида "28.04 (апреля).2006, день - 5 (пятница) 14:50:39".

Народ советует

Часто используемые форматы вывода даты и времени можно оформить в виде констант (в смысле, псевдоконстант, о которых говорилось в начале этой главы). Например:

```
var JSPS_DATETIMEFORMAT_FULL = "DA.MO.YE HO:MI:SE";
// 28.04.2006 14:50:39
var JSPS_DATETIMEFORMAT_DATELONG = "DA MN YE г.";
// 28 апреля 2006 г.
var JSPS_DATETIMEFORMAT_DATESHORT = "DA.MO.YE";
// 28.04.2006
```

```
var JSPS_DATETIMEFORMAT_TIMELONG = "HO:MI:SE";
// 14:50:39
var JSPS_DATETIMEFORMAT_TIMEMIDDLE = "HO:MI AM";
// 2:50 PM
var JSPS_DATETIMEFORMAT_TIMESHORT = "HO:MI";
// 14:50
```

Впоследствии нам будет достаточно передать методы `toFormattedString` объекта `Date` одну из этих констант-переменных:

```
var d = new Date();
var s = d.toFormattedString(JSPS_DATETIMEFORMAT_DATELONG);
```

Вычисление значения даты, отличающегося от заданного на определенное количество дней

Проблема

Я пишу сценарий, манипулирующий значениями даты. Мне нужно вычислить дату, отстоящую от заданной на определенное количество дней. Как это сделать?

Решение

Использование метода `addDays` (листинг 1.22) объекта `Date`. Формат его вызова таков:

```
<Дата>.addDays(<Количество дней>);
```

Единственным параметром в этот метод передается количество дней, на которые нужно увеличить значение даты, содержащееся в экземпляре объекта `Date`. (Чтобы уменьшить значение даты на определенное количество дней, нужно передать методу `addDays` отрицательное число, задающее это количество.) Метод не возвращает никакого значения.

Листинг 1.22. Метод `addDays` объекта `Date`, вычисляющий значение даты, отстоящее от заданной на определенное количество дней

```
function mjspsAddDays(daysCount)
  {
    var msInDay = 1000 * 3600 * 24;
    var msNow = this.getTime()
    this.setTime(msNow + msInDay * daysCount);
  }

Date.prototype.addDays = mjspsAddDays;
```

Примеры

```
var d = new Date();
var n = 20;
d.addDays(n);
document.write(d.toLocaleString());
```

Этот сценарий поместит на Web-страницу дату, отстоящую от текущей на 20 дней вперед.

```
var d = new Date();
var n = -10;
d.addDays(n);
document.write(d.toLocaleString());
```

А этот сценарий выведет дату, отстоящую от текущей на 10 дней назад.

Вычисление значения времени, отличающегося от заданного на определенное количество часов, минут и секунд

Проблема

Я пишу сценарий, манипулирующий значениями даты и времени. Мне нужно вычислить время, отстоящее от заданного на определенное количество часов, минут и секунд. Как это сделать?

Решение

Использование метода `addHMS` (листинг 1.23) объекта `Date`. Формат его вызова приведен далее:

```
<Дата>.addHMS(<Количество часов>, <Количество минут>,
↳<Количество секунд>);
```

Параметры этого метода задают количество часов, минут и секунд, на которые нужно увеличить значение даты и времени, содержащееся в экземпляре объекта `Date`. (Чтобы уменьшить значение времени на определенное количество часов, минут и секунд, нужно передать методу `addHMS` отрицательные числа, задающие эти количества.) Метод не возвращает никакого значения.

Листинг 1.23. Метод `addHMS` объекта `Date`, вычисляющий значение времени, отстоящее от заданного на определенное количество часов, минут и секунд

```
function mjspsAddHMS(hoursCount, minutesCount, secondsCount)
  {
    var msInSecond = 1000;
```

```
    var msInMinute = msInSecond * 60;
    var msInHour = msInMinute * 60;
    var msNow = this.getTime()
    this.setTime(msNow + msInHour * hoursCount + msInMinute *
    ↳minutesCount + msInSecond * secondsCount);
  }

Date.prototype.addHMS = mjspsAddHMS;
```

Примеры

```
var d = new Date();
var nh = 1;
var nm = 20;
var ns = 43;
d.addHMS(nh, nm, ns);
document.write(d.toLocaleString());
```

Этот сценарий выведет на Web-страницу значение даты и времени, отстоящее от текущего на 1 час, 20 минут и 43 секунды.

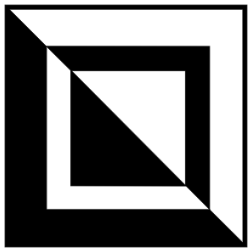
```
var d = new Date();
d.addHMS(0, -5, 0);
```

А этот сценарий "отодвинет" значение даты и времени, содержащееся в переменной `d`, на 5 минут в прошлое.

Что дальше?

Разобравшись с языком JavaScript и обогатив его новыми возможностями, перейдем к написанию Web-сценариев. В следующей главе народ посоветует, как удобнее обратиться к различным элементам Web-страницы, как лучше написать сценарий и как привязать обработчик к событию. А еще народ рассмотрит замечательную возможность Internet Explorer — поведения — и посетует, что остальные программы Web-обозревателей до сих пор их не поддерживают.

ГЛАВА 2



Написание Web-сценариев и обработка событий

В этой главе мы рассмотрим народные советы по написанию Web-сценариев и обработке событий. Также мы поговорим о создании поведений Microsoft Internet Explorer — замечательного инструмента, позволяющего привязать однотипные обработчики событий сразу к нескольким элементам Web-страницы.

Получение доступа к Web-странице и ее элементам

Начнем давать народные советы с того, что расскажем о различных способах получения доступа к нужному элементу Web-страницы из сценариев.

Как получить доступ к нужному элементу Web-страницы?

Проблема

Как можно из сценария получить доступ к свойствам и методам какого-либо элемента страницы?

Решение 1

Прежде всего, элемент страницы, к которому нужно получить доступ, должен иметь уникальное имя. Это имя задается с помощью атрибута `ID`, поддерживаемого всеми видимыми тегами. Например (атрибут, задающий имя элемента, выделен полужирным шрифтом):

```
<P ID="par">Текстовый абзац</P>
```

Народ замечает

Старые программы Web-обозревателей, в частности Navigator версий 4.*x*, не поддерживали атрибут `ID`. Вместо него использовался атрибут `NAME`. Поэтому для некоторых элементов Web-страниц (обычно это формы и элементы управления) имена ради совместимости до сих пор иногда задаются с использованием обоих этих атрибутов:

```
<P ID="par" NAME="par">Текстовый абзац</P>
```

Хотя это и не обязательно, т. к. рыночная доля старых Web-обозревателей исчезающе мала.

А имена фреймов задаются исключительно атрибутом `NAME`. Атрибут `ID` они не поддерживают до сих пор. (О фреймах разговор пойдет в *главе 9*.)

Задав имя, можно воспользоваться одной из коллекций объекта `document`. Этих коллекций довольно много, и все они перечислены в табл. 2.1.

Таблица 2.1. *Коллекции объекта* `document`

Коллекция	Описание
`all`	Все элементы страницы, включая теги <HTML>, <HEAD>, <TITLE> и <BODY>
`anchors`	Все якоря страницы, созданные с помощью тега <A NAME="*Имя якоря*">...</A>
`applets`	Все Java-апплеты, графические изображения и элементы ActiveX
`embeds`	Все мультимедийные элементы, помещенные в страницу с помощью тега <EMBED> (расширения Web-обозревателя)
`forms`	Все Web-формы
`frames`	Все фреймы
`images`	Все графические изображения
`links`	Все гиперссылки, включая "горячие" области (теги <AREA>)
`scripts`	Все Web-сценарии
`styleSheets`	Все таблицы стилей. Поддерживается только Internet Explorer, начиная с версии 4.0, и Firefox

Каждая из перечисленных в табл. 2.1 коллекций содержит все элементы страницы определенного типа. Так, коллекция `images` содержит все графические изображения, помещенные на страницу, а коллекция `links` — все гиперссылки. Ну, а самая большая коллекция `all` содержит вообще все элементы Web-страницы: текстовые абзацы, изображения, гиперссылки, таблицы, контейнеры и пр.

Например, так можно обратиться к абзацу с именем par, HTML-код которого был приведен ранее, и получить его содержимое в формате HTML с помощью свойства innerHTML:

```
var s = document.all["par"].innerHTML;
```

> ***Народ советует***
>
> Именно коллекцией all объекта document из-за ее всеобъемлющего характера пользоваться удобнее всего.

Примеры

```
<P ID="par">Это пример доступа к элементу Web-страницы.</P>
<SCRIPT TYPE="text/javascript">
  var parObject = document.all["par"];
  var s = parObject.innerHTML;
</SCRIPT>
```

Этот сценарий поместит в переменную s содержимое абзаца с именем par (текст "Это пример доступа к элементу Web-страницы."). Доступ к этому абзацу выполняется с помощью коллекции all объекта document.

> ***Внимание!***
>
> Приведенный пример работает только в Microsoft Internet Explorer и Opera. Firefox не поддерживает свойство innerHTML.

```
<IMG ID="pic" SRC="pic.gif">
<SCRIPT TYPE="text/javascript">
  var picObject = document.images["pic"];
  picObject.src = "pic2.gif";
</SCRIPT>
```

А этот сценарий заменит графическое изображение, сформированное тегом <IMG> с именем pic, на другое, хранящееся в файле pic2.gif.

Решение 2

Использование метода getElementById объекта document. Формат его вызова таков:

```
document.getElementById(<Имя элемента>);
```

Имя элемента задается в строковом виде (разумеется, оно должно быть задано в HTML-коде с помощью атрибута ID). Метод возвращает экземпляр объекта, соответствующий искомому элементу.

Это решение не очень наглядно, но зато правильно с точки зрения стандартов *DOM* (Document Object Model, объектная модель документа), установленных

комитетом *W3C*. (W3C, он же *WWWC*, он же *World Wide Web Consortium* — Комитет повсеместно протянутой паутины — орган, занимающийся интернет-стандартами.) Но используется оно довольно редко.

Кстати, нам также может пригодиться метод `getElementsByTagName`, поддерживаемый всеми объектами — элементами страницы. Он принимает единственный параметр — имя тега в строковом виде без символов < и > — и возвращает массив, содержащий экземпляры объектов, соответствующие элементам страницы, созданным с помощью данного тега.

Например, сценарий:

```
var tableObject = document.all["table1"];
var rowsArray = tableObject.getElementsByTagName("TR");
```

поместит в переменную `rowsArray` все строки таблицы `table1`.

Пример

Перепишем один из приведенных ранее примеров:

```
<IMG ID="pic" SRC="pic.gif">
<SCRIPT TYPE="text/javascript">
  var picObject = document.getElementById("pic");
  picObject.src = "pic2.gif";
</SCRIPT>
```

Как получить доступ к телу Web-страницы?

Проблема

Мне нужно получить доступ к телу Web-страницы, т. е. к тегу `<BODY>`. Есть ли для этого какие-то средства?

Решение

Воспользоваться свойством `body` объекта `document`. Это свойство возвращает экземпляр объекта, соответствующий телу Web-страницы.

Пример

```
var s = document.body.innerHTML;
```

Этот сценарий поместит в переменную `s` все содержимое страницы в виде HTML-кода.

> ***Внимание!***
>
> Приведенный пример работает только в Microsoft Internet Explorer и Opera. Firefox не поддерживает свойство `innerHTML`.

В каком месте страницы поместить Web-сценарий?

Проблема

Я не знаю, в каком месте HTML-кода страницы мне поместить свой Web-сценарий. Подскажите, пожалуйста!

Решение

Вообще, правило здесь достаточно простое — Web-сценарии прочитываются и выполняются Web-обозревателем в том месте HTML-кода, где они находятся, и в порядке от начала страницы к ее концу (сверху вниз, если смотреть на листинг HTML-кода страницы) — именно в таком порядке Web-обозреватель загружает Web-страницу. А вот где именно поместить тот или иной сценарий, зависит от его назначения.

- Сценарии, содержащие объявления глобальных переменных и функций и выполняющие какие-либо подготовительные действия, лучше помещать в секцию заголовка Web-страницы (в тег `<HEAD>`). Такие сценарии будут обработаны Web-обозревателем в самую первую очередь, еще до загрузки тела Web-страницы (содержимого тега `<BODY>`).
- Сценарии, выводящие какой-либо текст на Web-страницу, нужно поместить в соответствующие места секции тела страницы (тега `<BODY>`).
- Сценарии, которые должны быть выполнены после загрузки всего содержимого страницы, лучше всего поместить в самый конец ее HTML-кода. Зачастую их помещают даже после закрывающего тега `</HTML>`. Также эти сценарии можно оформить в виде обработчика события `onLoad` тега `<BODY>`. (Об обработчиках событий и самих событиях мы поговорим далее в этой главе.)

> ***Народ советует***
>
> Пожалуй, лучше и оформлять их в виде обработчика события `onLoad` тега `<BODY>`. Так делают чаще всего.

Очень часто на многих Web-страницах используются одни и те же функции и переменные (и псевдоконстанты, о которых шла речь в *главе 1*). Разумеется, объявления всех этих функций и переменных дублируются в коде всех страниц, где они используются. А это сильно увеличивает размер файлов страниц и усложняет техническую поддержку (например, чтобы исправить ошибку в одной функции, придется просматривать все ее объявления на всех использующих ее страницах).

В качестве выхода можно посоветовать поместить все объявления функций и псевдоконстант в отдельный *файл сценариев*. Файлы сценариев имеют тек-

стовый формат (как и файлы HTML) и расширение js. Они должны содержать только сам код JavaScript без тегов <SCRIPT>.

Чтобы использовать объявленные в файле сценариев функции и переменные в Web-странице, этот файл нужно сначала к ней подключить. Выполняется это с помощью такого вот тега:

```
<SCRIPT SRC="<Имя файла сценария>"></SCRIPT>
```

Этот тег практически всегда помещается в секции заголовка страницы (в теге <HEAD>). В конце концов, файлы сценариев практически всегда содержат объявления функций и псевдоконстант, которые должны быть загружены и обработаны в самую первую очередь. А именно секция заголовка страницы грузится самой первой.

Народ замечает

Файлы сценариев имеют еще одно преимущество. Как известно, Web-обозреватели сохраняют на жестком диске (кэшируют) все загруженные из Интернета файлы. Впоследствии, если какой-либо файл на Web-сервере не изменился, Web-обозреватель, чтобы не загружать его повторно, открывает с жесткого диска его кэшированную копию, что ощутимо увеличивает производительность. А, как правило, файлы сценариев, в отличие от Web-страниц, изменяются крайне редко, поэтому Web-обозреватель практически всегда для работы использует их кэшированные копии.

Работа со сценариями — обработчиками событий

Настала пора поговорить о сценариях-обработчиках событий, принципах их написания и привязки к элементам страницы и событиям.

Как создать сценарий — обработчик события?

Проблема

Мне нужно написать сценарий — обработчик события. Какие существуют способы привязать его к нужному мне элементу страницы и событию?

Решение 1

Проще и удобнее всего привязать обработчик прямо к тегу, создающему элемент страницы, событие которого нужно обработать. Вот синтаксис такой привязки:

```
<<Тег элемента страницы>
↳<Название события> = "<Код JavaScript обработчика события>"
↳[Остальные атрибуты тега]>
```

Фактически в этом случае код JavaScript сценария — обработчика события присваивается особому атрибуту тега, создающего элемент страницы. Имя этого атрибута совпадает с названием события, к которому нужно привязать обработчик. Например, событию `onClick`, наступающему при щелчке мышью на элементе страницы, соответствует атрибут `ONCLICK`:

```
<INPUT TYPE="button" ID="btnOK"
↳ONCLICK="<Код обработчика события onClick>">
```

Примеры

Обработчик события `onClick` кнопки (его код JavaScript выделен полужирным шрифтом):

```
<FORM>
  <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!"
  ↳ONCLICK="document.all['btnOK'].value = 'Нажато';">
</FORM>
```

Поскольку код обработчика совсем мал (одно-единственное выражение), мы присвоили его атрибуту `ONCLICK` тега `<INPUT>`. Этот атрибут, как мы уже знаем, соответствует событию `onClick`, наступающему при щелчке мышью.

Но если код обработчика события достаточно велик, лучше всего оформить его в виде функции, поместить ее объявление в секцию заголовка страницы, а в значение соответствующего событию атрибута поместить вызов этой функции. Давайте рассмотрим пример такой функции — обработчика события:

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function btnOKOnClick()
      {
        document.all["btnOK"].value = "Нажато";
        window.alert("Вы только что нажали кнопку!");
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM>
    <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!"
    ↳ONCLICK="btnOKOnClick();">
  </FORM>
</BODY>
```

Метод `alert` объекта `window` выводит на экран окно-предупреждение с текстом, переданным в качестве единственного параметра этого метода.

Решение 2

Использовать особые свойства экземпляра объекта, соответствующего нужному элементу страницы. Эти свойства имеют те же имена, что и соответствующие им события, и должны принимать в качестве значения функции-обработчики этих событий. Общий синтаксис написания этих обработчиков таков:

```
<Тег, создающий элемент страницы, события которого должны быть
↳обработаны>
<SCRIPT TYPE="text/javascript">
  <Объявление функции-обработчика>
  <Элемент страницы>.<Свойство, соответствующее событию> =
  ↳<Функция-обработчик>
</SCRIPT>
```

Обратим внимание — выражение, присваивающее функцию-обработчик соответствующим свойствам элемента страницы, должно помещаться после тега, создающего этот элемент.

Что касается свойств, которым нужно присвоить функции-обработчики, то они носят "говорящие" названия, как и описанные выше атрибуты. Так, знакомому нам событию `onClick` соответствует свойство `onclick`.

> ***Народ предупреждает***
>
> Имена свойств, соответствующих событиям, должны быть набраны только маленькими (строчными) буквами.

Пример

```
<BODY>
  <FORM>
    <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!">
  </FORM>
  <SCRIPT TYPE="text/javascript">
    function btnOKOnClick()
      {
        document.all["btnOK"].value = "Нажато";
      }
      document.all["btnOK"].onclick = btnOKOnClick;
  </SCRIPT>
</BODY>
```

Решение 3 (для Firefox)

Использование *функций-слушателей* DOM. Самый сложный способ, но, судя по всему, за ним будущее...

Первый шаг, который нам нужно сделать для создания обработчика события, — это написать функцию, которая, собственно, и станет этим самым обработчиком. Здесь нет ничего сложного — такие функции мы уже писали ранее, когда рассматривали предыдущие решения проблемы обработчиков событий.

Народ замечает

Функция-слушатель DOM может принимать один параметр. С этим параметром в функцию передается особый объект, предоставляющий сведения о событии. Мы рассмотрим его использование далее в этой главе.

Второй шаг, который мы сделаем, — зарегистрируем написанную функцию в качестве слушателя. Для этого используется метод `addEventListener` объекта, представляющего нужный элемент страницы. Формат вызова этого метода таков:

```
<Элемент страницы>.addEventListener(<Событие>, <Функция-слушатель>,
↳<Перехватывать события, наступающие во вложенных элементах>);
```

Первым параметром методу передается название события в виде *строки DOM*. Оно совпадает с привычным для нас названием события, но без символов "on" и набранное маленькими буквами. Со вторым параметром все понятно — он задает функцию-слушателя, которую мы только что объявили. А вот о третьем параметре нужно поговорить подробнее.

Дело в том, что функция-слушатель, привязанная к элементу страницы, при нашем желании может обрабатывать события, наступающие в элементах страницы, вложенных в данный элемент. При этом, если к какому-либо из вложенных элементов привязана другая функция-слушатель, она будет выполнена после того, как отработает функция-слушатель текущего элемента. Таким образом, текущий элемент страницы может перехватывать события, наступающие во вложенных элементах, даже раньше, чем эти самые вложенные элементы на них отреагируют. Кстати, такое поведение функции-слушателя и называется *перехватом событий*.

Так вот третий параметр метода и задает режим перехвата событий функцией-слушателем. Он имеет логический тип; значение `true` запускает перехват событий вложенных элементов, а значение `false` — отключает.

Метод `addEventListener` не возвращает никакого значения. Сразу же после его вызова, если третьим параметром было передано значение `true`, функция-слушатель начинает обработку всех указанных событий.

Народ замечает

Для элемента страницы можно зарегистрировать сколько угодно функций-слушателей. Можно даже дважды зарегистрировать одну функцию — с включенным и отключенным режимом перехвата событий.

Метод `removeEventListener` объекта, представляющего нужный элемент страницы, позволяет убрать зарегистрированную ранее функцию-слушателя. Формат его вызова таков:

```
<Элемент страницы>.removeEventListener(<Событие>, <Функция-слушатель>,
↳<Был ли включен режим перехвата событий>);
```

С первыми двумя параметрами все понятно. Третьим же параметром должно быть передано логическое значение `true` или `false`, в зависимости от того, был ли при регистрации функции-слушателя включен режим перехвата событий. Дело в том, что, как мы уже узнали, для элемента страницы можно зарегистрировать сколько угодно функций-слушателей, в том числе дважды зарегистрировать одну функцию — с включенным и отключенным режимом перехвата событий. Так вот, если мы так сделаем, нам придется убирать отдельно регистрацию с включенным режимом перехвата и отдельно регистрацию с отключенным режимом перехвата.

Метод `removeEventListener` также не возвращает никакого значения.

Еще один полезный метод — `dispatchEvent` — позволит нам перенаправить событие другому элементу страницы, чтобы он его обработал (если, конечно, для него зарегистрирована хоть одна функция-слушатель). Вот как он вызывается:

```
<Элемент страницы, которому перенаправляется событие>.dispatchEvent
↳(<Событие>);
```

Событие, передаваемое единственным параметром этого метода, — это единственный необязательный параметр функции-слушателя.

> ***Народ советует***
>
> С объектом, передаваемым этим параметром, в функцию-слушатель поступает много полезной информации о событии. Так что лучше всегда его принимать, т. е. указывать этот параметр в объявлении функции-слушателя.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function btnOKOnClick()
      {
        var btnOKObject = document.all["btnOK"];
        if (btnOKObject.value == "Нажми!")
          btnOKObject.value = "Нажато"
        else
          btnOKObject.value = "Нажми!";
      }
  </SCRIPT>
</HEAD>
```

```
<BODY>
  <FORM>
    <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!">
  </FORM>
  <SCRIPT TYPE="text/javascript">
      var btnOKObject = document.all["btnOK"];
      btnOKObject.addEventListener("click", btnOKOnClick, false);
  </SCRIPT>
</BODY>
```

Обратим внимание, что код, выполняющий регистрацию функции-слушателя для кнопки `btnOK`, мы поместили после тега, создающего эту кнопку. Если бы мы поместили его перед этим тегом, Web-обозреватель не смог бы найти кнопку, поскольку ее HTML-код еще не загружен, и выдал бы сообщение об ошибке.

Список доступных событий

Проблема

Я не могу запомнить все поддерживаемые Web-обозревателем события. Приведите, пожалуйста, их список.

Решение

Нет ничего проще — см. табл. 2.2.

Таблица 2.2. *События, доступные для обработки*

Событие	Строка DOM для Firefox	Описание
`onAbort`	`"abort"`	Наступает при прерывании загрузки Web-страницы, графического изображения, апплета Java, расширения Web-обозревателя или элемента ActiveX
`onBlur`	`"blur"`	Наступает, когда элемент управления теряет фокус ввода
`onChange`	`"change"`	Наступает, когда элемент управления теряет фокус ввода, в случае если его содержимое было изменено пользователем
`onClick`	`"click"`	Наступает при щелчке мышью на элементе страницы
`onContextMenu`		Наступает, когда посетитель щелкает по странице или одному из ее элементов правой кнопкой мыши, чтобы вывести контекстное меню, перед собственно выводом контекстного меню. Поддерживается только Microsoft Internet Explorer, начиная с версии 5.0

Таблица 2.2 *(продолжение)*

Событие	Строка DOM для Firefox	Описание
onDblClick		Наступает при двойном щелчке мышью на элементе страницы
onError	"error"	Наступает при неудачной загрузке графического изображения, апплета Java, расширения Web-обозревателя или элемента ActiveX или ошибке в Web-сценарии
onFocus	"focus"	Наступает, когда элемент управления получает фокус ввода
onHelp		Наступает, когда посетитель нажимает клавишу <F1>, чтобы вывести на экран интерактивную справку, перед собственно выводом справки. Поддерживается только Microsoft Internet Explorer, начиная с версии 4.0
onKeyDown		Наступает, когда посетитель нажимает и удерживает клавишу
onKeyPress		Наступает, когда посетитель нажимает клавишу
onKeyUp		Наступает, когда посетитель отпускает нажатую ранее клавишу
onLoad	"load"	Наступает после завершения загрузки страницы, графического изображения, апплета Java, расширения Web-обозревателя или элемента ActiveX
onMouseDown	"mousedown"	Наступает при нажатии кнопки мыши, когда ее курсор находится над элементом страницы, перед событиями onMouseUp и onClick
onMouseMove	"mousemove"	Наступает при перемещении курсора мыши над элементом страницы
onMouseOut	"mouseout"	Наступает, когда курсор мыши перемещается за границы элемента страницы
onMouseOver	"mouseover"	Наступает, когда курсор мыши помещается на элемент страницы
onMouseUp	"mouseup"	Наступает при отпускании кнопки мыши, когда ее курсор находится над элементом страницы, после события onMouseDown и перед событием onClick
onReset	"reset"	Наступает при очистке Web-формы
onResize	"resize"	Наступает при изменении размеров окна Web-обозревателя пользователем
onScroll	"scroll"	Наступает при прокрутке Web-страницы
onSelect	"select"	Наступает при выделении текста в поле ввода или области редактирования

Таблица 2.2 *(окончание)*

Событие	Строка DOM для Firefox	Описание
onSelectStart		Наступает, когда пользователь начинает выделять какой-либо текст на странице или в одном из ее элементов. Поддерживается только Microsoft Internet Explorer, начиная с версии 4.0
onSubmit	"submit"	Наступает при отправке данных Web-формой
onUnload	"unload"	Наступает после выгрузки Web-страницы

События, для которых не указана строка DOM, не могут быть обработаны с помощью функций-слушателей.

Получение информации о событии

Проблема

Мне нужно получить кое-какую информацию о наступившем событии, в частности, координаты точки, по которой пользователь щелкнул мышью. Как это сделать?

Решение (для Microsoft Internet Explorer и Opera)

Использовать соответствующие свойства объекта `event`, доступного в теле функции — обработчика события. Все они перечислены в табл. 2.3.

Таблица 2.3. *Свойства объекта* `event`

Свойство	Описание
altKey	Возвращает `true`, если была нажата клавиша <Alt>
altLeft	Возвращает `true`, если была нажата левая клавиша <Alt>, и `false`, если правая
button	Возвращает номер кнопки мыши, нажатой посетителем. Список возвращаемых значений приведен в табл. 2.4
cancelBubble	Задает, прерывать или не прерывать "всплытие" событий. О "всплытии" событий мы поговорим далее в этой главе
clientX	Возвращает горизонтальную координату курсора мыши относительно клиентской области окна Web-обозревателя (без учета рамок, заголовка, строки меню, панелей инструментов и строки состояния)

Таблица 2.3 *(окончание)*

Свойство	Описание
clientY	Возвращает вертикальную координату курсора мыши относительно клиентской области окна Web-обозревателя (без учета рамок, заголовка, строки меню, панелей инструментом и строки состояния)
ctrlKey	Возвращает true, если была нажата клавиша <Ctrl>
ctrlLeft	Возвращает true, если была нажата левая клавиша <Ctrl>, и false, если правая
fromElement	Возвращает элемент страницы, с которого переместился курсор мыши при наступлении события onMouseOver или onMouseOut
keyCode	Возвращает код нажатой клавиши в стандарте UNICODE
offsetX	Возвращает горизонтальную координату курсора мыши относительно элемента страницы, в котором наступило это событие
offsetY	Возвращает вертикальную координату курсора мыши относительно элемента страницы, в котором наступило это событие
repeat	Возвращает true, если событие onKeyPress наступило повторно вследствие того, что пользователь удерживает клавишу нажатой, и false в противном случае
returnValue	Разрешает или отменяет действие по умолчанию для элемента страницы. Подробнее об этом мы поговорим далее в этой главе
screenX	Возвращает горизонтальную координату курсора мыши относительно экрана
screenY	Возвращает вертикальную координату курсора мыши относительно экрана
shiftKey	Возвращает true, если была нажата клавиша <Shift>
shiftLeft	Возвращает true, если была нажата левая клавиша <Shift>, и false, если правая
srcElement	Возвращает элемент страницы, в котором наступило данное событие
toElement	Возвращает элемент страницы, на который был перемещен курсор мыши при наступлении события onMouseOver или onMouseOut
type	Возвращает имя события, набранное маленькими буквами, без символов "on"
x	Возвращает горизонтальную координату курсора мыши относительно родителя ("внешнего" по отношению к текущему элемента страницы)
y	Возвращает вертикальную координату курсора мыши относительно родителя

Таблица 2.4. *Значения, возвращаемые свойством* `button` *объекта* `event`

Значение	Описание
0	Ни одна из кнопок мыши не была нажата
1	Была нажата левая кнопка
2	Была нажата правая кнопка
3	Были одновременно нажаты левая и правая кнопки
4	Была нажата средняя кнопка
5	Были одновременно нажаты левая и средняя кнопки
6	Были одновременно нажаты правая и средняя кнопки
7	Были одновременно нажаты все кнопки

Пример

Далее приведен фрагмент кода Web-страницы, выводящей при щелчке мышью на ней окно-сообщение. В этом окне отображаются координаты курсора мыши относительно клиентской области окна Web-обозревателя.

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function bodyOnClick()
      {
        window.alert("x=" + event.clientX.toString() + ", y=" +
        ↳event.clientY.toString());
      }
  </SCRIPT>
</HEAD>
<BODY ONCLICK="bodyOnClick();">
  <P>Это некий текст.</P>
</BODY>
```

Решение (для Firefox)

Использовать соответствующие свойства объекта, который передается в функцию-слушатель и несет различную информацию о событии. Все они перечислены в табл. 2.5.

Таблица 2.5. *Свойства объекта, несущего информацию о событии*

Свойство	Описание
`altKey`	Возвращает `true`, если была нажата клавиша <Alt>
`bubbles`	Возвращает `true`, если событие может "всплывать". О "всплытии" событий будет рассказано далее в этой главе

Таблица 2.5 *(окончание)*

Свойство	Описание
button	Возвращает номер кнопки мыши, нажатой пользователем. Список возвращаемых значений приведен в табл. 2.4
cancelable	Возвращает true, если имеется возможность отменить действие по умолчанию для элемента страницы. О действиях по умолчанию и отмене их будет рассказано далее в этой главе
clientX	Возвращает горизонтальную координату курсора мыши относительно клиентской области окна Web-обозревателя (без учета рамок, заголовка, строки меню, панелей инструментов и строки состояния)
clientY	Возвращает вертикальную координату курсора мыши относительно клиентской области окна Web-обозревателя (без учета рамок, заголовка, строки меню, панелей инструментов и строки состояния)
ctrlKey	Возвращает true, если была нажата клавиша <Ctrl>
currentTarget	Возвращает элемент страницы, в котором в данный момент происходит обработка событий. Может не совпадать с элементом страницы, в котором наступило событие, если включен перехват событий *(см. ранее)* или событие "всплыло" *(см. далее)*
relatedTarget	Возвращает элемент страницы, с которого переместился курсор мыши при наступлении события onMouseOver, или элемент страницы, на который был перемещен курсор мыши при наступлении события onMouseOut
screenX	Возвращает горизонтальную координату курсора мыши относительно экрана
screenY	Возвращает вертикальную координату курсора мыши относительно экрана
shiftKey	Возвращает true, если была нажата клавиша <Shift>
target	Возвращает элемент страницы, в котором наступило событие
type	Возвращает имя события в виде строки DOM

Один обработчик событий сразу для нескольких элементов страницы

Проблема

Я хочу выполнять обработку событий сразу в нескольких элементах страницы одинаковым образом. Как это сделать?

Решение 1

Ну, проще всего привязать один и тот же обработчик ко всем этим элементам страницы.

Пример

Вот фрагмент HTML-кода Web-страницы, в которой один обработчик события `onClick` привязан сразу к трем кнопкам:

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    // Объявление "универсального" обработчика событий
    function eventHandler()
      {
        // Тело "универсального" обработчика события
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM>
    <INPUT TYPE="button" ID="btn1" VALUE="Кнопка 1"
    ↳ONCLICK="eventHandler();">
    <INPUT TYPE="button" ID="btn2" VALUE="Кнопка 2"
    ↳ONCLICK="eventHandler();">
    <INPUT TYPE="button" ID="btn3" VALUE="Кнопка 3"
    ↳ONCLICK="eventHandler();">
  </FORM>
</BODY>
```

Решение 2

Использовать одну интересную возможность, называемую *"всплытием"* событий. Она часто позволяет избежать очень многих проблем.

Рассказать о ней проще всего на примере. Предположим, что на нашей Web-странице имеется форма с кнопкой. При этом к телу страницы (т. е. к тегу `<BODY>`) и кнопке привязаны обработчик события `onClick`. Что они делают, для нас в данный момент не важно.

Теперь предположим, что пользователь щелкнул по кнопке. В кнопке наступит событие `onClick`. Web-обозреватель запустит обработчик того события, что привязан к кнопке, а обработчик события что-то там выполнит.

Вы думаете, что на этом все закончится? Нет. Web-обозреватель передаст событие элементу страницы, в который вложена кнопка, т. е. в форму. Если бы к форме был привязан обработчик этого события, он бы выполнился, но, поскольку мы этим не озаботились, форма на него никак не отреагирует.

Далее Web-обозреватель перенаправит событие в элемент страницы, в который вложена форма, т. е. в тело страницы. И обработчик события `onClick`, что привязан к форме, будет выполнен. И, поскольку тело страницы — самый верхний элемент в иерархии, здесь событие прекратит свое существование.

Событие как бы "всплывает" от элемента страницы, в котором оно наступило, в самый верхний элемент иерархии — тело страницы. При этом все обработчики этого события во всех элементах страницы, по которым "следует" событие, будут выполнены.

Пример

Вот фрагмент HTML-кода Web-страницы, в которой используется единый обработчик события `onClick` для всех кнопок:

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    // Объявление "универсального" обработчика событий
    function eventHandler()
      {
        // Тело "универсального" обработчика события
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM ONCLICK="eventHandler();">
    <INPUT TYPE="button" ID="btn1" VALUE="Кнопка 1">
    <INPUT TYPE="button" ID="btn2" VALUE="Кнопка 2">
    <INPUT TYPE="button" ID="btn3" VALUE="Кнопка 3">
  </FORM>
</BODY>
```

Как из обработчика события получить доступ к элементу страницы, в котором наступило это событие?

Проблема

Я хочу привязать одну и ту же функцию-обработчик к событиям сразу нескольких элементов управления. Как мне получить доступ к элементу страницы, в котором наступило событие, из тела функции-обработчика?

Решение 1

Самое очевидное и универсальное решение — это передать в функцию-обработчик экземпляр объекта, соответствующий элементу страницы, в кото-

ром произошло событие. Проще всего это сделать, использовав ключевое слово `this`.

> ***Народ предупреждает!***
>
> Ключевое слово `this` в данном случае имеет смысл только в коде JavaScript, что присвоен атрибуту тега элемента, соответствующему событию, в качестве значения. Использовать это ключевое слово в теле функции-обработчика нельзя — это вызовет ошибку.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function btnOKOnClick(sender)
      {
        sender.value = "Нажато";
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM>
    <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!"
    ↳ONCLICK="btnOKOnClick(this);">
  </FORM>
</BODY>
```

Решение 2 (для Microsoft Internet Explorer и Opera)

Использовать свойство `srcElement` описанного выше объекта `event`.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function btnOKOnClick()
      {
        event.srcElement.value = "Нажато";
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM>
    <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!"
    ↳ONCLICK="btnOKOnClick();">
```

```
    <INPUT TYPE="button" ID="btnCancel" VALUE="Меня тоже нажми!"
    ↳ONCLICK="btnOKOnClick();">
  </FORM>
</BODY>
```

Решение 2 (для Firefox)

Использовать свойство `target` описанного выше объекта, который передается в функцию-слушатель и несет различную информацию о событии.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function btnOKOnClick(event)
      {
        if (event.target.value == "Нажми!")
          event.target.value = "Нажато"
        else
          event.target.value = "Нажми!";
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM>
    <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!"
    ↳ONCLICK="btnOKOnClick();">
  </FORM>
  <SCRIPT TYPE="text/javascript">
    var btnOKObject = document.all["btnOK"];
    btnOKObject.addEventListener("click", btnOKOnClick, false);
  </SCRIPT>
</BODY>
```

Как прервать "всплытие" события?

Проблема

Мне нужно в одном из обработчиков прервать "всплытие" события. Можно ли это сделать?

Решение (для Microsoft Internet Explorer и Opera)

Присвоить значение `true` свойству `cancelBubble` описанного ранее объекта `event`.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function eventHandler()
      {
        window.alert("Кнопка нажата!");
      }

    function btn3OnClick()
      {
        window.alert("Кнопка 3 нажата!");
        event.cancelBubble = true;
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM ONCLICK="eventHandler();">
    <INPUT TYPE="button" ID="btn1" VALUE="Кнопка 1">
    <INPUT TYPE="button" ID="btn2" VALUE="Кнопка 2">
    <INPUT TYPE="button" ID="btn3" VALUE="Кнопка 3"
    ↳ONCLICK="btn3OnClick();">
  </FORM>
</BODY>
```

Решение (для Firefox)

Вызвать метод `stopPropagation` описанного ранее объекта, передаваемого в функцию-слушатель и несущего информацию о событии. Этот метод не принимает параметров и не возвращает значения.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function eventHandler()
      {
        window.alert("Кнопка нажата!");
      }

    function btn3OnClick(event)
      {
        window.alert("Кнопка 3 нажата!");
        event.stopPropagation();
      }
```

```
  </SCRIPT>
</HEAD>
<BODY>
  <FORM ID="frm">
    <INPUT TYPE="button" ID="btn1" VALUE="Кнопка 1">
    <INPUT TYPE="button" ID="btn2" VALUE="Кнопка 2">
    <INPUT TYPE="button" ID="btn3" VALUE="Кнопка 3">
  </FORM>
  <SCRIPT TYPE="text/javascript">
    var frmObject = document.all["frm"];
    frmObject.addEventListener("click", eventHandler, false);
    var btn3Object = document.all["btn3"];
    btn3Object.addEventListener("click", btn3OnClick, false);
  </SCRIPT>
</BODY>
```

Отмена действия по умолчанию в ответ на событие

Проблема

Мне необходимо отменить действие по умолчанию, которое Web-обозреватель выполняет в ответ на событие (переход по гиперссылке при щелчке на ней, отправку данных Web-формой при нажатии кнопки **Отправить** и пр.). Как это сделать?

Решение 1

Вставить в сценарий, являющийся значением соответствующего событию атрибута тега элемента, такое выражение:

```
return false;
```

Это выражение должно быть самым последним в теле функции-обработчика.

Примеры

```
<P><A HREF="page1.html" ONCLICK="return false;">Страница 1</A></P>
```

При щелчке на этой гиперссылке перехода по указанному в атрибуте `HREF` тега `<A>` интернет-адресу не произойдет.

```
<A HREF="page1.html" ONCLICK="onClickHandler(); return false;">
↳Страница 1</A>
```

А при щелчке на этой гиперссылке выполнится объявленная ранее функция `onClickHandler`. Перехода по указанному в атрибуте `HREF` тега `<A>` интернет-адресу также не произойдет.

Решение 2 (для Microsoft Internet Explorer и Opera)

Присвоить значение `false` свойству `returnValue` описанного ранее объекта `event`.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function onClickHandler()
      {
        // Остальной код тела функции-обработчика
        event.returnValue = false;
      }
  </SCRIPT>
</HEAD>
<BODY>
  <P><A HREF="page1.html" ONCLICK="onClickHandler();">Страница 1</A></P>
</BODY>
```

При щелчке на гиперссылке перехода по указанному в атрибуте `HREF` тега `<A>` интернет-адресу не произойдет.

Решение 2 (для Firefox)

Вызвать метод `preventDefault` описанного ранее объекта, передаваемого в функцию-слушатель и несущего информацию о событии. Этот метод не принимает параметров и не возвращает значения.

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    function hrf1OnClick(event)
      {
        // Остальной код тела функции-обработчика
        event.preventDefault();
      }
  </SCRIPT>
</HEAD>
<BODY>
  <P><A HREF="page1.html" ID="hrf1">Страница 1</A></P>
  <SCRIPT TYPE="text/javascript">
    var hrf1Object = document.all["hrf1"];
    hrf1Object.addEventListener("click", hrf1OnClick, false);
  </SCRIPT>
</BODY>
```

При щелчке на гиперссылке перехода по указанному в атрибуте HREF тега <A> интернет-адресу не произойдет.

Перехват обработки событий

Проблема

Мне нужно вклиниться в обработку событий, т. е. подменить уже привязанный к событию обработчик своим с возможностью выполнять и "старый" обработчик. Как это можно сделать?

Решение

1. Объявляем переменную, в которой будет храниться "старый" обработчик, и присваиваем ей значение null.
2. Проверяем, присвоен ли свойству элемента страницы, соответствующему нужному нам событию, обработчик события, и, если присвоен, помещаем его в объявленную ранее переменную.
3. Объявляем функцию, которая станет "новым" обработчиком, и присваиваем ее нужному свойству элемента страницы.
4. Если нам понадобится выполнить "старый" обработчик, обращаемся к переменной, где он хранится, но перед этим убеждаемся, что ей не присвоено значение null.

> ***Народ предупреждает!***
>
> Этот прием имеет смысл использовать, только если обработчик события привязан через атрибут тега элемента страницы или аналогичное свойство представляющего его экземпляра объекта. Если же событие обрабатывается функцией-слушателем, он не будет работать. (Хотя в этом случае можно просто зарегистрировать для нужного события элемента страницы еще одну функцию-слушатель, которая будет работать вместе с уже зарегистрированными.)

Пример

```
<HEAD>
  <SCRIPT TYPE="text/javascript">
    // Объявляем переменную для хранения "старого" обработчика
    var oldbtnOKClickHandler = null;

    // Объявляем функцию, которая станет "новым" обработчиком
    function btnOKOnClick()
      {
        // Код "нового" обработчика
        window.alert("Кнопка нажата!");
```

```
        // Проверяем, существовал ли "старый" обработчик, и, если
        // существовал, выполняем его
        if (oldbtnOKClickHandler) oldbtnOKClickHandler();
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM>
    // Создаем кнопку btnOK, к событию onClick
    // которой привязан обработчик.
    // Мы назвали его "старым"
    <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!"
    ↳ONCLICK="document.all['btnOK'].value = 'Нажато';">
  </FORM>
  <SCRIPT TYPE="text/javascript">
    // Получаем доступ к кнопке btnOK
    var btnOKObject = document.all["btnOK"];

    // Проверяем, был ли к событию onClick уже привязан "старый"
    // обработчик, и, если был привязан, присваиваем его объявленной
    // в самом начале примера переменной
    if (btnOKObject.onclick) oldbtnOKClickHandler = btnOKObject.onclick;

    // Привязываем к событию onClick кнопки btnOK "новый" обработчик
    btnOKObject.onclick = btnOKOnClick;
  </SCRIPT>
</BODY>
```

Работа с поведениями Microsoft Internet Explorer

Поведения Microsoft Internet Explorer, которые начали поддерживаться в версии 5.5 этой программы, — очень мощный инструмент создания динамических Web-страниц. К сожалению, он плохо описан в популярных книгах по Web-дизайну и Web-программированию, поэтому используется нечасто.

Народ замечает

А может, все дело в том, что поведения поддерживаются только Internet Explorer. Остальные Web-обозреватели их не поддерживают. Наверно, все это потому, что их разработчики не рассматривают свои творения как платформы для создания полноценных Web-решений корпоративного уровня, где поведения могут весьма пригодиться.

Народ предупреждает!

Напоминаем еще раз, что поведения поддерживаются ТОЛЬКО Internet Explorer версии 5.5 или более поздними.

Что такое поведения Microsoft Internet Explorer и как их создавать?

Проблема

Вот все говорят: "Поведения Internet Explorer... Поведения Internet Explorer..." А что они собой представляют и как их создавать? Наверно, очень сложно...

Решение

Поведения Microsoft Internet Explorer — это аналог каскадных таблиц стилей CSS, но не для правил оформления страниц, а для Web-сценариев. Если совсем просто, то поведение — это текстовый файл с расширением htc и содержащий код сценариев — обработчиков событий, которые могут быть привязаны к любому элементу страницы так же, как обычные стили CSS.

Создать поведение так же просто, как таблицу стилей. Сначала мы привязываем к нужному нам элементу страницы стилевой класс. Делается это хорошо нам знакомым атрибутом `CLASS`:

```
<<Тег> CLASS="<Имя стилевого класса>" [<Остальные атрибуты тега>]>
```

Следующий шаг — определение этого стилевого класса в таблице стилей, внешней или внутренней. Это тоже нам прекрасно знакомо.

Итак, стилевой класс создан, атрибуты стиля и их значения написаны. Теперь нам нужно указать, что мы также собираемся привязать к этому стилю еще и поведение. (Сам файл поведения мы пока не создали, но создадим потом.) Делается это с помощью вот такого атрибута стиля:

```
behavior:url(<Имя файла поведения>);
```

Как видим, атрибут этот имеет "говорящее" имя — behavior ("поведение"). Также заметим, что *Имя файла поведения* указывается без кавычек.

Народ замечает

В принципе, городить огород с таблицей стилей совсем не обязательно. Атрибут стиля `behavior` можно поместить и во встроенный стиль, привязанный к тегу с помощью атрибута `STYLE` этого тега:

```
<<Тег> STYLE="behavior:url(<Имя файла поведения>);
↳[Остальные атрибуты стиля]" [<Остальные атрибуты тега>]>
```

Третий, и последний, шаг — создать сам файл поведения. Как мы уже выяснили, он должен быть текстовым и иметь расширение htc. Формат его содержимого должен быть таким:

```
<PUBLIC:COMPONENT>
  <Определение обработчиков событий>
  <SCRIPT TYPE="text/javascript">
    <Код обработчиков событий>
  </SCRIPT>
</PUBLIC:COMPONENT>
```

Весь код, описывающий поведение, должен помещаться внутри парного тега <PUBLIC:COMPONENT>. Код JavaScript обработчиков событий также должен быть заключен в парный тег <SCRIPT>.

Что касается *Определений обработчиков событий*, то они должны иметь такой формат:

```
<PUBLIC:ATTACH EVENT="<Имя события>" [FOR="<Элемент страницы>"]
↳ONEVENT="<Вызов функции-обработчика события>"/>
```

Значение обязательного атрибута EVENT одинарного тега <PUBLIC:ATTACH> должно содержать имя события в виде строки, набранной маленькими буквами. Значение другого обязательного атрибута — ONEVENT — должно содержать вызов функции-обработчика этого события, объявленной ниже, в теге <SCRIPT>. Здесь все понятно.

Что касается необязательного атрибута FOR, то он задает элемент страницы или объект Web-обозревателя, событие которого должно обрабатываться. Всего он может принимать три значения:

- "element" — элемент страницы, к которому было привязано поведение. Это значение по умолчанию, применяемое, если атрибут FOR не указан;
- "document" — тело Web-страницы;
- "window" — окно Web-обозревателя.

И еще один важный момент. Мы привыкли, что в языке HTML одинарные теги не содержат закрывающей пары. Но поведения — другое дело, и даже одинарный тег (например, <PUBLIC:ATTACH>) должен содержать закрывающий символ / перед символом >. Вот так:

```
<PUBLIC:ATTACH <Атрибуты> />
```

Впрочем, пробел перед этим символом необязателен.

Пример

Давайте создадим поведение, запрещающее реакцию Web-обозревателя при щелчке по элементам Web-страницы. Впоследствии оно может нам пригодиться.

Итак, вот содержимое файла jspsIEBehaviorSimple.html с самой Web-страницей:

```
<HTML>
  <HEAD>
    <STYLE TYPE="text/css">
      .nodefault { behavior:url(jspsIEBehaviorSimple.htc); }
    </STYLE>
  </HEAD>
  <BODY>
    <P><A HREF="page1.html" CLASS="nodefault">Страница 1</A></P>
    <P><A HREF="page2.html" CLASS="nodefault">Страница 2</A></P>
    <P><A HREF="page3.html">Страница 3</A></P>
  </BODY>
</HTML>
```

А вот содержимое файла jspsIEBehaviorSimple.htc с кодом поведения:

```
<PUBLIC:COMPONENT>
  <PUBLIC:ATTACH EVENT="onclick" ONEVENT="elementOnClick();"/>
  <SCRIPT TYPE="text/javascript">
    function elementOnClick()
      {
        event.returnValue = false;
      }
  </SCRIPT>
</PUBLIC:COMPONENT>
```

Если открыть файл jspsIEBehaviorSimple.html в Web-обозревателе, то при щелчках по гиперссылкам, ведущим на страницы 1 и 2, ничего происходить не будет (т. е. наше поведение работает). А гиперссылка, указывающая на страницу 3, работать будет, поскольку мы не указали в ее теге стилевой класс `nodefault`, к которому привязано поведение.

Как из поведения получить доступ к элементу страницы, к которому оно привязано, и телу страницы?

Проблема

Поведения, конечно, хорошая штука... Но мне нужно из него получить доступ к элементу страницы, к которому оно привязано, и к телу самой страницы. Это возможно?

Решение

Разумеется! Для этой цели Internet Explorer предоставляет программисту два объекта:

- `element` — элемент страницы, к которому привязано поведение;
- `document` — тело самой этой страницы.

Эти объекты доступны из любого сценария, являющегося частью кода поведения.

Пример

Давайте немного исправим содержимое файла jspsIEBehaviorSimple.htc, приведенное ранее (добавленный код выделен полужирным шрифтом):

```
<PUBLIC:COMPONENT>
  <PUBLIC:ATTACH EVENT="onclick" ONEVENT="elementOnClick();"/>
  <SCRIPT TYPE="text/javascript">
    function elementOnClick()
      {
        window.alert(element.innerText + " недоступна");
        event.returnValue = false;
      }
  </SCRIPT>
</PUBLIC:COMPONENT>
```

Теперь при щелчке на любой гиперссылке, к которой привязано это поведение, на экран будет выводиться окно-предупреждение, сообщающее, что данная страница недоступна.

> ***Народ замечает***
>
> Вот, кстати, одно из преимуществ поведений Internet Explorer. Чтобы исправить какой-либо из обработчиков событий, нам достаточно внести правки в один-единственный файл — файл поведения.

Как из поведения отследить момент окончания загрузки элемента страницы, к которому привязано поведение, и тела страницы?

Проблема

Хорошо, вы меня убедили! Я таки написал свое поведение, но столкнулся с проблемой. Мне нужно отследить момент, когда Web-обозреватель заканчивает загрузку элемента страницы, к которому привязано поведение, и тела самой страницы. Как это можно сделать?

Решение

Что ж, разработчики из Microsoft и это продумали. Internet Explorer предоставляет аж два события, доступных из кода поведения и сообщающих об окончании загрузки:

- элемента страницы, к которому привязано поведение, — onContentReady;
- тела страницы — onDocumentReady.

Обработать эти события проще простого. В начале кода поведения помещаем уже знакомый нам одинарный тег <PUBLIC:ATTACH> такого вида:

```
<PUBLIC:ATTACH EVENT="oncontentready|ondocumentready"
[FOR="<Элемент страницы>"]
ONEVENT="<Вызов функции-обработчика события>"/>
```

И пишем объявление самой функции-обработчика.

> ***Народ замечает***
>
> Вообще, поведение может обрабатывать еще два события, о которых автор не счел нужным упомянуть. Это событие onContentSave, наступающее при сохранении Web-страницы на жестком диске и копировании ее в буфер обмена, и событие onDetach, наступающее, когда поведение отключается от элемента страницы (это происходит при выгрузке страницы). Иногда они бывают полезными.

Пример

Давайте сделаем так, чтобы гиперссылки, к которым было привязано наше поведение, сразу же после загрузки зачеркивались. Для этого перепишем файл поведения jspsIEBehaviorSimple.htc так:

```
<PUBLIC:COMPONENT>
  <PUBLIC:ATTACH EVENT="onclick" ONEVENT="elementOnClick();"/>
  <PUBLIC:ATTACH EVENT="oncontentready" ONEVENT="elementOnReady();"/>
  <SCRIPT TYPE="text/javascript">
    function elementOnClick()
      {
        event.returnValue = false;
      }

    function elementOnReady()
      {
        element.style.textDecoration = "line-through";
      }
  </SCRIPT>
</PUBLIC:COMPONENT>
```

Свойство style экземпляра объекта, соответствующего элементу страницы, предоставляет доступ к одноименному объекту — стилю CSS. Свойство textDecoration, как и атрибут стиля text-decoration, позволяет задать оформление текста (подчеркивание, надчеркивание или зачеркивание). Значение "line-through" этого атрибута как раз зачеркивает текст.

И в этом случае нам не придется переделывать страницу jspsIEBehaviorSimple.html. Вот она, мощь поведений!

Как создать новое свойство поведения?

Проблема

Конечно, вынесение обработки событий в поведения здóрово помогает в работе. А можно ли аналогичным образом добавить элементам страницы новые свойства, причем так, чтобы их значения можно было бы задавать прямо в теге, как его атрибут?

Решение

Конечно можно! Для этого используется одинарный тег <PUBLIC:PROPERTY>. Вот его формат:

```
<PUBLIC:PROPERTY NAME="<Имя свойства>" [VALUE="<Значение по умолчанию>"]
[INTERNALNAME="<Имя переменной, соответствующей свойству>"]
[GET="<Вызов get-функции>"] [PUT="<Вызов put-функции>"]
[ID=<Имя элемента поведения, описывающего свойство>]/>
```

Единственный обязательный атрибут тега <PUBLIC:PROPERTY> — NAME, в котором задается имя свойства. Встретив такой тег, Web-обозреватель сам сформирует в памяти переменную, имя которой совпадает с именем свойства, и поместит в него заданное в атрибуте тега или свойстве экземпляра объекта, соответствующего элементу страницы. Нам самим объявлять эту переменную не нужно.

> ***Народ предупреждает!***
>
> Если значение свойства задано с помощью соответствующего атрибута в теге элемента страницы, оно передается в свойство в строковом виде.

Необязательный атрибут VALUE позволяет задать значение свойства по умолчанию. Это значение свойство примет в том случае, если оно не было явно задано в атрибуте тега.

Часто бывает так, что выбранное имя свойства в коде JavaScript использовать невозможно. Это может случиться, если его имя совпадает с какой-либо переменной, функцией или объектом, уже объявленным в коде поведения. Специально для такого случая у нас имеется возможность сохранить значение свойства в другой переменной. Вот как это делается...

1. Объявляем глобальную переменную, в которой будет храниться значение свойства, и присваиваем ей значение свойства по умолчанию в строковом виде. Атрибут VALUE тега <PUBLIC:PROPERTY> в этом случае можно не указывать.

2. Указываем имя объявленной переменной в значении необязательного атрибута INTERNALNAME тега <PUBLIC:PROPERTY>.

Теперь для доступа к значению свойства мы можем использовать эту переменную.

Очень часто бывает так, что значение свойства не хранится в памяти постоянно, а получается в результате каких-либо вычислений. В таком случае мы поступим следующим образом.

1. Объявим функцию, которая будет выполнять соответствующие вычисления и возвращать их результат. Параметров эта функция принимать не должна.
2. Поместим имя (только имя, а не вызов!) этой функции в значение необязательного атрибута GET тега <PUBLIC:PROPERTY>.

Кстати, функции, занимающиеся вычислениями значения какого-либо свойства, часто называют *get-функциями*, от английского "get" — "получить".

Также часто приходится при занесении нового значения в свойство выполнять какие-либо сторонние действия (например, обновлять содержимое элемента страницы). В таком случае алгоритм наших действий приведен ниже.

1. Объявляем переменную, в которой будет храниться значение свойства, — это обязательно! Указывать ее имя в атрибуте INTERNALNAME тега <PUBLIC:PROPERTY> в этом случае не нужно.
2. Объявляем функцию, принимающую единственный параметр — новое значение свойства — и не возвращающую результата.
3. Даем имя тегу <PUBLIC:PROPERTY>, описывающему свойство. Это делается с помощью хорошо знакомого нам необязательного атрибута ID.
4. В теле функции пишем код, который заносит значения полученного параметра в объявленную ранее переменную и выполняет нужные сторонние действия.
5. Там же, в теле функции, записываем вызов метода fireChange экземпляра объекта, представляющего тег <PUBLIC:PROPERTY>, описывающий данное свойство. Получить доступ к этому экземпляру объекта можно через имя, которое мы дали с помощью атрибута ID. Метод fireChange не принимает параметров, не возвращает результат и служит для информирования Web-обозревателя о том, что значение свойства элемента страницы изменилось.
6. Записываем имя (опять же, только имя, а не вызов!) этой функции в значение необязательного атрибута PUT тега <PUBLIC:PROPERTY>.
7. Если хотим, чтобы свойство было доступно еще и для чтения, а не только для записи, создаем get-функцию. Как это сделать, было описано ранее.

Народ предупреждает!

Мы должны создать и get-функцию, и set-функцию, если хотим, чтобы свойство было доступно и для чтения, и для записи. Если задана только get-функция, свойство будет доступно только для чтения. Если же задана только set-функция, свойство станет доступным только для записи.

Если заданы get- и set-функции, значение атрибута `INTERNALNAME` игнорируется.

Функции, выполняющие сторонние действия, помимо занесения значения свойства в переменную, называются *set-функциями*, от английского "set" — "установить".

Пример 1

Давайте создадим для всех гиперссылок страницы jspsIEBehaviorSimple.html свойство `enabled`. Будучи установленным в `true` (это, кстати, значение по умолчанию), оно разрешает переход по гиперссылке при щелчке по ней. Значение `false`, напротив, запрещает это.

Вот содержимое обновленного файла jspsIEBehaviorSimple.html:

```
<HTML>
  <HEAD>
    <STYLE TYPE="text/css">
      A { behavior:url(jspsIEBehaviorSimple.htc); }
    </STYLE>
  </HEAD>
  <BODY>
    <P><A HREF="page1.html">Страница 1</A></P>
    <P><A HREF="page2.html">Страница 2</A></P>
    <P><A HREF="page3.html">Страница 3</A></P>
  </BODY>
</HTML>
```

Обратим внимание, что мы привязали поведение ко всем гиперссылкам, создав для них соответствующий стиль переопределения тега. Там нам будет проще в дальнейшем экспериментировать с гиперссылками.

А вот содержимое переписанного файла jspsIEBehaviorSimple.htc:

```
<PUBLIC:COMPONENT>
  <PUBLIC:ATTACH EVENT="onclick" ONEVENT="elementOnClick();"/>
  <PUBLIC:ATTACH EVENT="oncontentready" ONEVENT="elementOnReady();"/>
  <PUBLIC:PROPERTY NAME="enabled" INTERNALNAME="isEnabled"/>
  <SCRIPT TYPE="text/javascript">
    var isEnabled = "true";
```

```
    function elementOnClick()
      {
        event.returnValue = (isEnabled == "true");
      }

    function elementOnReady()
      {
        if (isEnabled == "true")
          element.style.textDecoration = "none"
        else
          element.style.textDecoration = "line-through";
      }
  </SCRIPT>
</PUBLIC:COMPONENT>
```

Откроем новый файл jspsIEBehaviorSimple.html в Web-обозревателе. Мы сразу же заметим, что все гиперссылки реагируют на щелчки мышью. Так и должно было быть, поскольку мы не указали в их тегах атрибуты `ENABLED`, соответствующие созданному нами свойству, значит, по умолчанию оно получит значение `"true"`.

Теперь давайте немного исправим этот файл, добавив в тег `<A>`, создающий первую гиперссылку, атрибут `ENABLED` со значением `"false"`:

```
<P><A HREF="page1.html" ENABLED="false">Страница 1</A></P>
```

Сохраним этот файл и снова загрузим его в Web-обозревателе. Теперь первая гиперссылка зачеркнута и не функционирует. Наше поведение работает!

Пример 2

Исправим немного только что созданное поведение таким образом, чтобы при присвоении свойству `enabled` экземпляра объекта, соответствующего гиперссылке, нового значения внешний вид этой гиперссылки соответственно изменялся. Для этого нам придется написать get- и set-функции.

Начнем исправления с файла jspsIEBehaviorSimple.htc:

```
<PUBLIC:COMPONENT>
  <PUBLIC:ATTACH EVENT="onclick" ONEVENT="elementOnClick();"/>
  <PUBLIC:PROPERTY NAME="enabled" GET="getIsEnabled" PUT="putIsEnabled"
  ↳ID="prpEnabled"/>
  <SCRIPT TYPE="text/javascript">
    var isEnabled = true;

    function elementOnClick()
      {
        event.returnValue = isEnabled;
      }
```

```
    function getIsEnabled()
      {
        return isEnabled.toString();
      }

    function putIsEnabled(propValue)
      {
        isEnabled = (propValue == "true");
        prpEnabled.fireChange();
        if (isEnabled)
          element.style.textDecoration = "none"
        else
          element.style.textDecoration = "line-through";
      }
  </SCRIPT>
</PUBLIC:COMPONENT>
```

Здесь мы внесли одно усовершенствование — стали хранить значение свойства в переменной `isEnabled` в логическом виде. Все-таки с логическими величинами управляться проще, чем со строковыми.

А вот исправленный файл страницы jspsIEBehaviorSimple.html, с помощью которого мы будем тестировать новое поведение:

```
<HTML>
  <HEAD>
    <STYLE TYPE="text/css">
      A { behavior:url(jspsIEBehaviorSimple.htc); }
    </STYLE>
    <SCRIPT TYPE="text/javascript">
      function toggleLinkEnabled()
        {
          var hrf1Object = document.all["hrf1"];
          if (hrf1Object.enabled == "true")
            hrf1Object.enabled = "false"
          else
            hrf1Object.enabled = "true";
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P><A HREF="page1.html" ID="hrf1">Страница 1</A></P>
    <P><A HREF="page2.html">Страница 2</A></P>
    <P><A HREF="#" ONCLICK="toggleLinkEnabled();">Переключить</A></P>
  </BODY>
</HTML>
```

Мы привязали к третьей гиперссылке обработчик события `onClick`, вызывающий функцию, которая то включает гиперссылку, делая ее работоспособной, то отключает. В качестве значения атрибута `HREF` тега `<A>` этой гиперссылки мы указали символ "решетки" (`#`), обозначающий *"пустую"* гиперссылку (не вызывающую переход по какому-либо интернет-адресу).

Что ж, осталось только проверить готовое поведение в работе. Откроем страницу jspsIEBehaviorSimple.html в Web-обозревателе и попробуем пощелкать по гиперссылкам. Должно работать.

Пример 3

Как-то не очень красиво мы сделали — значения созданному нами свойству можно присваивать только в строковом виде. Давайте немного исправим функцию `putIsEnabled` поведения, чтобы она могла обрабатывать также и другие типы значений. Исправленная функция будет выглядеть так:

```
function putIsEnabled(propValue)
  {
    switch (typeof(propValue))
      {
        case "string":
          isEnabled = (propValue == "true");
          break;
        case "number":
          isEnabled = (propValue > 0);
          break;
        case "boolean":
          isEnabled = propValue;
          break;
        case "object":
          isEnabled = (propValue != null);
          break;
        default:
          isEnabled = false;
      }
    prpEnabled.fireChange();
    if (isEnabled)
      element.style.textDecoration = "none"
    else
      element.style.textDecoration = "line-through";
  }
```

Остальной код поведения, как и код страницы, остался без изменений.

Как создать новый метод поведения?

Проблема

Что ж, поведения Internet Explorer действительно на высоте. Даже свойства позволяют создавать. А как насчет методов?

Решение

Для добавления нового метода нам, прежде всего, нужно будет объявить функцию, которая, собственно, и станет этим методом. Чтобы сделать ее "видимой снаружи", используется одинарный тег `<PUBLIC:METHOD>`. Вот его формат:

```
<PUBLIC:METHOD NAME="<Имя метода>"
↳[INTERNALNAME="<Внутреннее имя метода>"]/>
```

Единственный обязательный атрибут `NAME` задает имя метода, под которым к нему смогут получить доступ сценарии, созданные в Web-странице. Разумеется, имя метода должно совпадать с именем объявленной нами функции — реализации этого метода.

Если же имя написанной нами функции все же не совпадает с выбранным нами именем метода, мы можем воспользоваться необязательным атрибутом `INTERNALNAME`. В качестве значения этого атрибута указывается имя функции — реализации метода — без скобок.

Пример

Давайте добавим нашему поведению два метода — `enable` и `disable`. Первый метод будет включать гиперссылку, второй — отключать.

Вот вновь исправленный файл поведения jspsIEBehaviorSimple.htc:

```
<PUBLIC:COMPONENT>
  <PUBLIC:ATTACH EVENT="onclick" ONEVENT="elementOnClick();"/>
  <PUBLIC:PROPERTY NAME="enabled" GET="getIsEnabled" PUT="putIsEnabled"
  ↳ID="prpEnabled"/>
  <PUBLIC:METHOD NAME="enable" INTERNALNAME="elementEnable"/>
  <PUBLIC:METHOD NAME="disable" INTERNALNAME="elementDisable"/>
  <SCRIPT TYPE="text/javascript">
    var isEnabled = true;

    function elementOnClick()
      {
        event.returnValue = isEnabled;
      }
```

```
    function getIsEnabled()
      {
        return isEnabled.toString();
      }

    function putIsEnabled(propValue)
      {
        switch (typeof(propValue))
          {
            case "string":
              isEnabled = (propValue == "true");
              break;
            case "number":
              isEnabled = (propValue > 0);
              break;
            case "boolean":
              isEnabled = propValue;
              break;
            case "object":
              isEnabled = (propValue != null);
              break;
            default:
              isEnabled = false;
          }
        prpEnabled.fireChange();
        if (isEnabled)
          element.style.textDecoration = "none"
        else
          element.style.textDecoration = "line-through";
      }

    function elementEnable()
      {
        putIsEnabled("true");
      }

    function elementDisable()
      {
        putIsEnabled("false");
      }
  </SCRIPT>
</PUBLIC:COMPONENT>
```

И не забудем исправить Web-страницу jspsIEBehaviorSimple.html, которая поможет нам его проверить:

```
<HTML>
  <HEAD>
    <STYLE TYPE="text/css">
      A { behavior:url(jspsIEBehaviorSimple.htc); }
    </STYLE>
    <SCRIPT TYPE="text/javascript">
      function toggleLinkEnabled()
        {
          var hrf1Object = document.all["hrf1"];
          if (hrf1Object.enabled == "true")
            hrf1Object.disable()
          else
            hrf1Object.enable();
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P><A HREF="page1.html" ID="hrf1">Страница 1</A></P>
    <P><A HREF="page2.html">Страница 2</A></P>
    <P><A HREF="#" ONCLICK="toggleLinkEnabled();">Страница 3</A></P>
  </BODY>
</HTML>
```

Как создать новое событие поведения?

Проблема

Ну и не помешала бы возможность создавать в поведениях свои события. Это можно сделать?

Решение

И это можно сделать! Правда, и времени это займет больше, чем добавление свойств и методов.

Сначала нам будет нужно объявить само событие. Делается это с помощью одинарного тега `<PUBLIC:EVENT>`, формат которого очень прост:

```
<PUBLIC:EVENT NAME="<Имя события>"
↳[ID=<Имя элемента поведения, описывающего событие>]/>
```

Обязательный атрибут `NAME` задает имя события, под которым оно будет доступно в коде Web-страницы. Другой атрибут, который, хоть и не является

обязательным, но практически всегда необходим, — ID, задающий уникальное имя тега <PUBLIC:EVENT>. Зачем он здесь нужен, мы узнаем чуть позже.

Следующий наш шаг — инициирование наступления этого события. Прежде всего, находим в коде поведения место, где созданное нами событие должно быть инициировано, и освобождаем пространство, по крайней мере, для двух выражений. Эти выражения будут, соответственно, создавать объект event, несущий информацию о событии, и инициировать это событие.

Объект event создается с помощью функции createEventObject. Эта функция не принимает никаких параметров, а возвращает созданный объект event, который будет нужно сразу же присвоить какой-либо переменной.

Мы можем установить любые значения для любых свойств этого объекта. Более того, мы даже можем создать свои свойства, если нам потребуется передавать с событием какую-то дополнительную информацию.

Теперь, когда объект event готов, мы можем инициировать наступление события. Для этого предназначен метод fire экземпляра объекта, представляющего собой описывающий событие тег <PUBLIC:EVENT>. Вот формат его вызова:

```
<Имя элемента поведения, описывающего событие>.fire
(Экземпляр объекта event);
```

Думается, объяснять тут особо нечего — все и так понятно.

Пример 1

Давайте добавим нашему многофункциональному и многострадальному поведению событие onReject, наступающее, когда пользователь щелкает мышью на отключенной гиперссылке. Для этого мы опять перепишем файл jspsIEBehaviorSimple.htc. Вот фрагмент переписанного файла:

```
<PUBLIC:COMPONENT>
  <PUBLIC:ATTACH EVENT="onclick" ONEVENT="elementOnClick();"/>
  <PUBLIC:PROPERTY NAME="enabled" GET="getIsEnabled" PUT="putIsEnabled"
  ID="prpEnabled"/>
  <PUBLIC:METHOD NAME="enable" INTERNALNAME="elementEnable"/>
  <PUBLIC:METHOD NAME="disable" INTERNALNAME="elementDisable"/>
  <PUBLIC:EVENT NAME="onReject" ID="evnOnReject"/>
  <SCRIPT TYPE="text/javascript">
    var isEnabled = true;

    function elementOnClick()
      {
        event.returnValue = isEnabled;
```

```
        if (!isEnabled)
          {
            var evn = createEventObject();
            evnOnReject.fire(evn);
          }
      }

    // Остальной код поведения был опущен для экономии места
  </SCRIPT>
</PUBLIC:COMPONENT>
```

И, в какой уже раз, перепишем страницу jspsIEBehaviorSimple.html:

```
<HTML>
  <HEAD>
    <STYLE TYPE="text/css">
      A { behavior:url(jspsIEBehaviorSimple.htc); }
    </STYLE>
    <SCRIPT TYPE="text/javascript">
      function hrfOnReject()
        {
          window.alert("Эта гиперссылка недоступна");
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P><A HREF="page1.html" ENABLED="false"
    ↳ONREJECT="hrfOnReject();">Страница 1</A></P>
    <P><A HREF="page2.html" ONREJECT="hrfOnReject();">Страница 2</A></P>
    <P><A HREF="page3.html" ONREJECT="hrfOnReject();">Страница 3</A></P>
  </BODY>
</HTML>
```

Пример 2

Дабы еще усовершенствовать наше поведение, давайте передавать в значении одного из свойств объекта `event` интернет-адрес отключенной гиперссылки. Назовем это свойство `href`.

Чтобы реализовать эту возможность в нашем поведении, нам будет достаточно переписать в файле jspsIEBehaviorSimple.htc только функцию `elementOnClick`. Вот так она должна теперь выглядеть:

```
function elementOnClick()
  {
    event.returnValue = isEnabled;
```

```
    if (!isEnabled)
      {
        var evn = createEventObject();
        evn.href = element.href;
        evnOnReject.fire(evn);
      }
  }
```

Остальной код поведения остался без изменений.

И еще мы обязательно внесем изменения в функцию-обработчик события `hrfOnReject` страницы jspsIEBehaviorSimple.html. Исправленная функция будет выглядеть так:

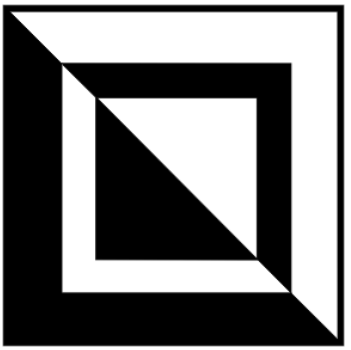
```
function hrfOnReject()
  {
    window.alert("Гиперссылка " + event.href + " недоступна");
  }
```

Остальной код страницы остался без изменений.

> ***Народ советует***
>
> Что касается поведений, то автор о многом умолчал. Если же вы хотите знать о них все, посетите раздел сайта *MSDN* (Microsoft Developer's Network, сеть разработчиков Microsoft), посвященный поведениям Internet Explorer. Найти этот раздел можно по адресу **http://msdn.microsoft.com/workshop/author/behaviors/behaviors_node_entry.asp**.

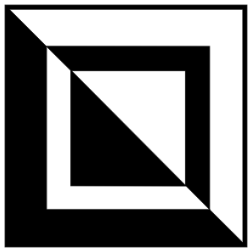
Что дальше?

Выяснив все о сценариях, событиях и их обработке, перейдем к решению конкретных задач. И начнем мы с получения сведений о клиентской операционной системе и Web-обозревателя, которым пользуется посетитель нашего будущего сайта. Такая информация очень часто бывает нужна, например, чтобы подогнать страницу под конкретное разрешение экрана или выдать посетителю страницу на определенном языке. Итак, приступим!



ЧАСТЬ II

Работа с Web-обозревателем



ГЛАВА 3

Получение сведений о клиенте

В этой главе представлены народные советы по работе с Web-обозревателем. И в первую очередь — о получении сведений о самой программе Web-обозревателя и компьютере пользователя (т. е. клиента). А сведения эти самые разнообразные: название и версии Web-обозревателя, поддерживаемые языки, разрешение и цветность экрана и пр.

Как выяснить разрешение и цветность экрана на компьютере клиента?

Проблема

Мне нужно динамически изменять страницу в зависимости от параметров экрана клиентского компьютера. Как получить эти параметры?

Решение

Использовать объект `screen`, свойства которого (табл. 3.1) представляют доступ к параметрам видеоподсистемы клиентского компьютера.

Таблица 3.1. *Свойства объекта* `screen`

Свойство	Описание
`availHeight`	Возвращает высоту полезной области экрана без панели задач и прочих панелей
`availWidth`	Возвращает ширину полезной области экрана без панели задач и прочих панелей
`colorDepth`	Возвращает цветность. Для 16 цветов возвращается значение 2, для 256 — 8, для 16,7 миллионов цветов (режим HiColor) — 32

Таблица 3.1 *(окончание)*

Свойство	Описание
height	Возвращает полную высоту экрана
width	Возвращает полную ширину экрана

Народ советует

Лучше использовать для выяснения разрешения экрана свойства availWidth и availHeight, а не width и height. Последние два свойства не учитывают, что системная панель задач у пользователя может быть настроена так, чтобы присутствовать на экране постоянно, в результате полезная площадь экрана уменьшится.

Народ предупреждает!

Web-обозреватель Opera здесь отличился — в нем свойства screenLeft и screenTop возвращают координаты окна относительно экрана, а позиционируется окно относительно главного окна программы. Поэтому придется от значения горизонтальной координаты отнять 10 (толщина рамок главного окна), а от значения координаты вертикальной — 90 пикселов (рамка главного окна плюс его заголовок со строкой меню и панелями инструментов). Эти значения выяснены автором книги экспериментально.

Пример

Вот небольшая Web-страничка, выдающая разрешение экрана клиентского компьютера и его цветность.

```
<HTML>
  <HEAD>
    <TITLE>Разрешение экрана</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      document.write("<P>Разрешение экрана вашего компьютера - " +
      ↳screen.width.toString() + "x" + screen.height.toString() +
      ↳".</P>");
      document.write("<P>В том числе, полезное - " +
      ↳screen.availWidth.toString() + "x" +
      ↳screen.availHeight.toString() + ".</P>");
      document.write("<P>Цветность - " + screen.colorDepth.toString() +
      ↳" бит.</P>");
    </SCRIPT>
  </BODY>
</HTML>
```

Как получить сведения о Web-обозревателе?

Проблема

Я хочу вывести на Web-страницу сведения о Web-обозревателе клиента, в частности, его название и версию. Можно ли мне их выяснить и, если можно, то как?

Решение

Воспользоваться объектом `navigator`. Все свойства этого объекта вместе с их описаниями приведены в табл. 3.2.

Таблица 3.2. *Свойства объекта* `navigator`

Свойство	Описание
`appCodeName`	Возвращает имя исходного кода программного ядра Web-обозревателя. Полное описание возвращаемого этим свойством значения приведено далее в тексте книги
`appName`	Возвращает имя программы Web-обозревателя. Полное описание возвращаемого этим свойством значения приведено далее в тексте книги
`appVersion`	Возвращает версию программы Web-обозревателя. Полное описание возвращаемого этим свойством значения приведено далее в тексте книги
`browserLanguage`	Возвращает код языка программы Web-обозревателя (например, "ru" для русского языка, "en" для английского и пр.). Поддерживается только Internet Explorer, начиная с версии 4.0, и Opera
`cookieEnabled`	Возвращает `true`, если Web-обозревателю разрешен пользователем прием cookie. Поддерживается только Internet Explorer, начиная с версии 4.0
`cpuClass`	Возвращает класс процессора клиентского компьютера, например, "x86" или "Alpha". Поддерживается только Internet Explorer, начиная с версии 4.0
`language`	Возвращает код языка программы Web-обозревателя. Поддерживается только Firefox и Opera
`onLine`	Возвращает `true`, если клиент в настоящий момент подключен к Интернету (находится в режиме on-line), и `false`, если отключен от него (off-line). Поддерживается только Internet Explorer, начиная с версии 4.0, и Firefox
`platform`	Возвращает обозначение операционной системы клиента, например, "Win32"

Таблица 3.2 *(окончание)*

Свойство	Описание
systemLanguage	Возвращает код языка операционной системы клиента. Поддерживается только Internet Explorer, начиная с версии 4.0
userAgent	Возвращает строку, идентифицирующую Web-обозреватель клиента. Полное описание возвращаемого этим свойством значения приведено далее в тексте книги
userLanguage	То же самое, что browserLanguage. Поддерживается только Internet Explorer, начиная с версии 4.0, и Opera

Объект navigator поддерживает, кроме того, метод javaEnabled(), возвращающий true, если Web-обозреватель может выполнять Web-сценарии JavaScript. Этот метод весьма полезен, поскольку пользователь может отключить исполнение Web-сценариев в настройках безопасности Web-обозревателя.

А теперь нужно дать пояснения по поводу некоторых из приведенных в табл. 3.2 свойств. Всего этих свойств четыре: appCodeName, appName, appVersion и userAgent.

Начнем со свойства appCodeName. Для всех Web-обозревателей оно вернет строку "Mozilla" — название старой программы, на исходном коде которой основаны все современные Web-обозреватели. Современный Web-обозреватель Mozilla (ныне — SeaMonkey) — это, можно сказать, дальнейшее развитие старого Mozilla.

Фактически свойство appCodeName — дань памяти современных Web-обозревателей "старичкам". Но для нас оно совершенно бесполезно.

Со свойствами appName, appVersion и userAgent все много сложнее. Для каждой из рассмотренных в книге программ возвращаемые ими значения будут разными.

Начнем со свойства appName. Возвращаемые им строковые значения будут такими:

- "Microsoft Internet Explorer" для Internet Explorer и Opera в режиме совместимости с Internet Explorer 6.0;
- "Netscape" для Mozilla, Firefox, Navigator и Opera в режиме совместимости с Mozilla 5.0;
- "Opera" для Opera в режиме представления "своим именем".

Народ замечает

Обратим внимание, что для Mozilla, Firefox и Opera в режиме совместимости с Mozilla 5.0 значение свойства appName будет равно "Netscape". Дело в том, что

Web-обозреватели Mozilla и Firefox основаны на исходных текстах Netscape Navigator.

Теперь обратимся к свойству `appVersion`. Для Internet Explorer возвращаемое этим свойством значение будет иметь такой вид:

```
<Версия исходного кода ядра> (compatible; <Версия>;
<Операционная система>)
```

На компьютере автора значение свойства `appVersion` таково:

```
4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
```

Народ поясняет

Windows NT 5.1 — это Windows XP. Windows 2000 будет обозначаться как Windows NT 5.0. Строка ".NET CLR 1.1.4322" обозначает версию установленной на компьютере исполняющей среды Microsoft .NET. Что обозначает строка "SV1", автору книги установить не удалось.

Перейдем к Opera. Если в ее настройках задан режим совместимости с Internet Explorer 6.0, значение свойства `appVersion` будет немного другим. Вот его формат:

```
<Версия исходного кода ядра> (compatible; MSIE 6.0;
<Операционная система>; <Язык>)
```

Например:

```
4.0 (compatible; MSIE 6.0; Windows NT 5.1; ru)
```

В случае если мы включим режим совместимости Opera с Mozilla 5.0 (имеются в виду Web-обозреватели, построенные на основе исходного кода старого Mozilla версии 5.0), получим значение, формат которого таков:

```
<Версия исходного кода ядра> (<Операционная система>; U|I; <Язык>)
```

Например:

```
5.0 (Windows NT 5.1; U; ru)
```

Народ замечает

Буква "U" в приведенном выше формате значения свойства `appVersion`, по идее, обозначает американскую версию программы, а буква "I" — интернациональную. В реальности же там почему-то всегда присутствует буква "U".

В случае если в настройках Opera задан режим "представления своим именем", мы получим такое значение:

```
<Версия Opera> (<Операционная система>; U|I; <Язык>)
```

Например:

```
8.54 (Windows NT 5.1; U; ru)
```

Лаконичные Mozilla, Firefox и Navigator не принесут нам таких сюрпризов, как многоликий (даже чересчур) Web-обозреватель Opera. Формат выдаваемого ими значения свойства `appVersion` таков:

<Версия исходного кода ядра> (*<Операционная система>*; *<Язык>*)

Автор получил такое значение для русской версии Firefox 1.5.0.3:

```
5.0 (Windows; ru)
```

для Mozilla 1.7 и Navigator 7 preview 1:

```
5.0 (Windows; en-US)
```

> ***Народ предупреждает!***
>
> Заметим, что Mozilla и Navigator выдают пятибуквенный код языка, например, "ru-RU" для русского или "en-US" для американского английского. Все остальные Web-обозреватели выдают двухбуквенный код языка, например, "ru" или "en".

Значение, возвращаемое свойством `userAgent`, также сильно зависит от Web-обозревателя. Internet Explorer, например, выдаст вот что:

<Значение свойства appCodeName>/*<Значение свойства appVersion>*

Например:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
↳1.1.4322)
```

Web-обозреватель Opera, "притворяющийся" Internet Explorer 6.0 или Mozilla 5.0, при обращении к свойству `userAgent` выдаст такую же строку:

<Значение свойства appCodeName>/*<Значение свойства appVersion>* Opera
↳*<Версия Opera>*

Например,

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; ru) Opera 8.54
```

в режиме совместимости с Internet Explorer 6.0 и

```
Mozilla/5.0 (Windows NT 5.1; U; ru) Opera 8.54
```

в режиме совместимости с Mozilla 5.0. Заметим, то в обоих случаях в конце строки указывается версия Opera, так что мы всегда сможем выяснить, Opera ли это.

Web-обозреватель Opera, "представляющийся самим собой", выдаст такое значение этого свойства:

<Значение свойства appCodeName>/*<Значение свойства appVersion>*

т. е. как и Internet Explorer. Например:

```
Opera/8.54 (Windows NT 5.1; U; ru)
```

Firefox, как обычно, идет своим путем. Значение свойства `userAgent` в его случае будет таким:

```
Mozilla/<Версия исходного кода ядра>
↳(<Название операционной системы>; U|I; <Версия операционной системы>;
↳<Язык>; <Ближайшая версия Mozilla>) <Версия программного ядра>
↳Firefox/<Версия Firefox>
```

Например:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.0.3) Gecko/20060426
↳Firefox/1.5.0.3
```

Народ поясняет

Gecko — это название программного ядра Firefox. На этом ядре также построен его "прародитель" Mozilla (ныне — SeaMonkey).

На очереди — Mozilla. Он в этом смысле мало отличается от своего "потомка" Firefox и выдаст вот что:

```
Mozilla/<Версия исходного кода ядра>
↳(<Название операционной системы>; U|I; <Версия операционной системы>;
↳<Язык>; <Версия Mozilla>) <Версия программного ядра>
```

Например:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040616
```

Что касается "ветерана" Navigator, то вот формат его значения свойства `userAgent`:

```
Mozilla/<Версия исходного кода ядра>
↳(<Название операционной системы>; U|I; <Версия операционной системы>;
↳<Язык>; <Ближайшая версия Mozilla>) <Версия программного ядра>
↳Netscape/<Версия Navigator>
```

На своем компьютере автор получил вот что:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0rc2) Gecko/20020512
↳Netscape/7.0b1
```

Народ замечает

Да, со свойствами `appVersion` и `userAgent` каши не сваришь... Мало того, что они возвращают довольно сложную строку, которую еще нужно разобрать, так еще и строка эта разная для разных программ Web-обозревателей (и даже для различных режимов работы в случае Opera). Хотя, так ли уж часто нам нужны версия программного ядра или точные сведения о .NET... Да и точная версия Web-обозревателя нужна не так уж и часто.

Пример

Вот простенькая Web-страничка, выводящая язык, поддерживаемый программой Web-обозревателя (русский или английский):

```
<HTML>
  <HEAD>
    <TITLE>Язык Web-обозревателя</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var langName = "";
      if (navigator.appName == "Microsoft Internet Explorer")
        langName = navigator.browserLanguage
      else
        langName = navigator.language;
      switch (langName.substring(0, 2))
        {
          case "ru":
            langDesc = "русский";
            break;
          case "en":
            langDesc = "английский";
            break;
          default:
            langDesc = "&lt;не установлен&gt;";
        }
      document.write("<P>Язык, поддерживаемый вашим
      ↳Web-обозревателем, -" + langDesc + ".</P>");
    </SCRIPT>
  </BODY>
</HTML>
```

Заметим, что язык проверяется по первым двум символам строки, возвращаемой свойством `browserLanguage` или `language`. Это нужно для поддержки Mozilla и Navigator, которые, как мы уже знаем, возвращают пятибуквенный код языка.

Как определить название и версию Web-обозревателя?

Проблема

Мне очень нужно узнать название программы Web-обозревателя и его точную версию. Помогите!

Решение

Использование универсальной функции `jspsGetProgramInfo` (листинг 3.1), поддерживающей Internet Explorer, Opera, Mozilla, Firefox и Navigator. Эта функция не принимает параметров и возвращает экземпляр особого объекта `ProgramInfo`, свойства которого предоставляют доступ к различным параметрам Web-обозревателя.

Свойства объекта `ProgramInfo` таковы:

- `programName` — название программы Web-обозревателя;
- `programVersion` — версия программы Web-обозревателя;
- `OperaCompatibilityMode` — режим совместимости Opera ("MSIE 6.0", "Mozilla 5.0" или "Opera"; для всех остальных Web-обозревателей возвращает пустую строку);
- `programPlatform` — полное название клиентской операционной системы;
- `programLanguage` — язык, поддерживаемый Web-обозревателем ("ru" для русского языка, "en" для английского и т. д.);
- `systemLanguage` — язык клиентской операционной системы (если Web-обозреватель не поддерживает этот параметр, возвращается пустая строка).

Все возвращаемые этими свойствами значения имеют строковый тип.

Листинг 3.1. Функция `jspsGetProgramInfo`, возвращающая сведения о Web-обозревателе

```
var JSPS_GPI_MSIE = "Microsoft Internet Explorer";
var JSPS_GPI_OPERA = "Opera";
var JSPS_GPI_MOZILLA = "Mozilla";
var JSPS_GPI_FIREFOX = "Firefox";
var JSPS_GPI_NAVIGATOR = "Navigator";

// Объявление объекта ProgramInfo
function ProgramInfo()
  {
    this.programName = "";
    this.programVersion = "";
    this.OperaCompatibilityMode = "";
    this.programPlatform = "";
    this.programLanguage = "";
    this.systemLanguage = "";
  }
```

```
// Объявление самой функции jspsGetProgramInfo
function jspsGetProgramInfo()
  {
    var piObj = new ProgramInfo();
    var ua = navigator.userAgent;
    var an = navigator.appName;
    var isntOpera = (ua.indexOf(JSPS_GPI_OPERA) == -1);
    var n1, n2;
    switch (an)
      {
        case JSPS_GPI_MSIE:
          piObj.programLanguage = navigator.browserLanguage;
          if (isntOpera)
            {
              piObj.programName = an;
              n1 = ua.indexOf("MSIE") + 5;
              n2 = ua.indexOf(";", n1);
              piObj.programVersion = ua.substring(n1, n2);
              n1 = n2 + 2;
              n2 = ua.indexOf(";", n1);
              piObj.programPlatform = ua.substring(n1, n2);
              piObj.systemLanguage = navigator.systemLanguage;
            }
          else
            {
              piObj.programName = JSPS_GPI_OPERA;
              n1 = ua.lastIndexOf(" ") + 1;
              n2 = ua.length;
              piObj.programVersion = ua.substring(n1, n2);
              piObj.OperaCompatibilityMode = "MSIE 6.0";
              n1 = ua.indexOf("MSIE") + 10;
              n2 = ua.indexOf(";", n1);
              piObj.programPlatform = ua.substring(n1, n2);
            }
          break;
        case "Netscape":
          if (isntOpera)
            {
              n1 = ua.indexOf(";") + 1;
              n1 = ua.indexOf(";", n1) + 2;
              n2 = ua.indexOf(";", n1);
              piObj.programPlatform = ua.substring(n1, n2);
```

```
        if (ua.indexOf(JSPS_GPI_FIREFOX) != -1)
          {
            piObj.programName = JSPS_GPI_FIREFOX;
            n1 = ua.lastIndexOf("/") + 1;
            n2 = ua.length;
            piObj.programVersion = ua.substring(n1, n2);
            piObj.programLanguage = navigator.language;
          }
        else
          {
            piObj.programLanguage =
            ↳navigator.language.substring(0, 2);
            if (ua.indexOf("Netscape") != -1)
              {
                piObj.programName = JSPS_GPI_NAVIGATOR;
                n1 = ua.lastIndexOf("/") + 1;
                n2 = ua.length;
                piObj.programVersion = ua.substring(n1, n2);
              }
            else
              {
                piObj.programName = JSPS_GPI_MOZILLA;
                n1 = ua.indexOf("rv:") + 3;
                n2 = ua.indexOf(")");
                piObj.programVersion = ua.substring(n1, n2);
              }
          }
      }
    else
      {
        piObj.programName = JSPS_GPI_OPERA;
        n1 = ua.lastIndexOf(" ") + 1;
        n2 = ua.length;
        piObj.programVersion = ua.substring(n1, n2);
        piObj.OperaCompatibilityMode = "Mozilla 5.0";
        n1 = ua.indexOf("(") + 1;
        n2 = ua.indexOf(";", n1);
        piObj.programPlatform = ua.substring(n1, n2);
        piObj.programLanguage = navigator.browserLanguage;
      }
    break;
  case JSPS_GPI_OPERA:
    piObj.programName = an;
```

```
            n1 = ua.indexOf("/") + 1;
            n2 = ua.indexOf(" ", n1);
            piObj.programVersion = ua.substring(n1, n2);
            piObj.OperaCompatibilityMode = an;
            n1 = n2 + 2;
            n2 = ua.indexOf(";", n1);
            piObj.programPlatform = ua.substring(n1, n2);
            piObj.programLanguage = navigator.browserLanguage;
            break;
        }
      return piObj;
    }
```

Хорошая идея!

Поместите объявление этой функции в файл сценариев browserdetect.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="browserdetect.js"></SCRIPT>
```

Кроме объявления функции `jspsGetProgramInfo` и служебного объекта `ProgramInfo`, листинг 3.1 содержит также объявления пяти псевдоконстант, содержащих названия всех Web-обозревателей, опознаваемых этой функцией. Вот эти константы:

- `JSPS_GPI_MSIE` содержит строку "Microsoft Internet Explorer";
- `JSPS_GPI_OPERA` содержит строку "Opera";
- `JSPS_GPI_MOZILLA` содержит строку "Mozilla";
- `JSPS_GPI_FIREFOX` содержит строку "Firefox";
- `JSPS_GPI_NAVIGATOR` содержит строку "Navigator".

Мы можем использовать их в сценариях для проверки, та ли это программа Web-обозревателя, которая нам нужна:

```
var pi = jspsGetProgramInfo();
if (pi.programName == JSPS_GPI_MSIE)
  . . .
```

Пример

Вот фрагмент кода Web-страницы, выдающей сведения о программе Web-обозревателя:

```
<BODY>
  <SCRIPT TYPE="text/javascript">
    var pi = jspsGetProgramInfo();
```

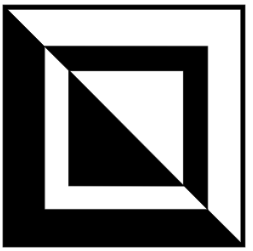
```
    document.write("<P>Название программы - " + pi.programName +
    ↳".</P>");
    document.write("<P>Версия программы - " + pi.programVersion +
    ↳".</P>");
    if (pi.OperaCompatibilityMode != "")
      document.write("<P>Режим совместимости с " +
      ↳pi.OperaCompatibilityMode + ".</P>");
    document.write("<P>ОС клиента - " + pi.programPlatform + ".</P>");
    document.write("<P>Язык программы - " + pi.programLanguage +
    ↳".</P>");
    if (pi.systemLanguage != "")
      document.write("<P>Язык ОС клиента - " + pi.systemLanguage +
      ↳".</P>");
  </SCRIPT>
</BODY>
```

Народ советует

Если нам понадобится писать сценарии, включающие многочисленные проверки Web-обозревателя, лучше всего вызвать функцию `jspsGetProgramInfo` в самом начале и поместить возвращенное ей значение в глобальную переменную. Впоследствии нам не придется вызывать каждый раз эту функцию — нужные данные всегда будут под рукой.

Что дальше?

На этом борьба с Web-обозревателями не закончена. В следующей главе народ посоветует, как управлять его окнами: открывать их, закрывать, перемещать и изменять их размеры. Также будет представлен мастер-класс написания обработчиков событий открытия, закрытия и перемещения этих самых окон. Так что держись, Web-обозреватель!



ГЛАВА 4

Управление Web-обозревателем

Настала пора, так сказать, активных действий по отношению к Web-обозревателю. В этой главе мы займемся открытием и закрытием его окон, изменением их размеров и перемещением. Также пойдет разговор о событиях, связанных с окнами, их обработчиках и о перенаправлении Web-обозревателя на другие страницы в зависимости от определенных условий.

Управление окнами Web-обозревателя

И начнем мы с народных советов по управлению окнами Web-обозревателя. Мы выясним, как их открывать, перемещать, изменять их размеры и закрывать.

Как открыть окно Web-обозревателя?

Проблема

Мне нужно при открытии моей страницы открыть другую страницу в другом окне Web-обозревателя программно (из Web-сценария). Как это сделать?

Решение

Использовать метод `open` объекта `window`. Вот формат его вызова:

```
window.open(<Интернет-адрес>, <Имя окна>,
[<Список параметров окна, разделенных запятыми>);
```

Первым параметром в этот метод передается *Интернет-адрес* Web-страницы, которая должна быть открыта в новом окне. Разумеется, *Интернет-адрес* должен быть задан в строковом виде. Если вместо него передана пустая строка, будет открыто пустое окно.

Второй параметр задает имя создаваемого окна в строковом виде. Это имя необходимо, если к созданному программно окну планируется обращаться с помощью гиперссылок (атрибут TARGET тега <A>) или Web-форм (атрибут TARGET тега <FORM>). Если же этого не требуется, вместо имени можно передать пустую строку.

Третьим, необязательным, параметром методу передается список различных параметров создаваемого окна, разделенных запятыми, в строковом виде. Все эти параметры перечислены в табл. 4.1.

Таблица 4.1. *Доступные параметры окна, передаваемые методу* `open`

Параметр	Описание
channelmode=yes|no	Если yes, то создаваемое окно будет отображаться с панелью каналов (так называемый "режим театра"). Поддерживается только Internet Explorer, начиная с версии 4.0
fullscreen=yes|no	Если yes, то создаваемое окно займет весь экран (так называемый "режим киоска"). Поддерживается только Internet Explorer, начиная с версии 4.0
height=<*Высота*>	Задает высоту создаваемого окна в пикселах
left=<*Координата X*>	Задает горизонтальную координату левого верхнего угла создаваемого окна. Поддерживается только Internet Explorer, начиная с версии 4.0
location=yes|no	Если yes, созданное окно будет содержать панель адреса
menubar=yes|no	Если yes, созданное окно будет содержать главное меню
replace=yes|no	Если yes, то интернет-адрес Web-страницы, открытой в создаваемом окне, заместит в списке истории интернет-адрес страницы, открытой в создающем окне. Поддерживается только Internet Explorer, начиная с версии 4.0
resizeable=yes|no	Если yes, размеры созданного окна можно будет изменять
scrollbars=yes|no	Если yes, созданное окно будет содержать полосы прокрутки
status=yes|no	Если yes, созданное окно будет содержать строку статуса
titlebar=yes|no	Если yes, созданное окно будет иметь заголовок. Поддерживается только Internet Explorer версий 4.*x* и 5.*x*
toolbar=yes|no	Если yes, созданное окно будет содержать панель инструментов
top=<*Координата Y*>	Задает вертикальную координату левого верхнего угла создаваемого окна. Поддерживается только Internet Explorer, начиная с версии 4.0
width=<*Ширина*>	Задает ширину создаваемого окна в пикселах

Метод `open` возвращает экземпляр объекта `window`, соответствующий открытому окну. С помощью этого экземпляра объекта мы можем управлять созданным окном.

Примеры

```
window.open("page.html", "second_window");
```

Это выражение откроет новое окно Web-обозревателя и выведет в нем страницу page.html. При этом создаваемое окно будет иметь случайные размеры и местоположение, а также содержать заголовок, строку меню, панель инструментов, панель адреса и строку состояния.

```
window.open("page.html", "second_window", "height=100, location=no,
↳menubar=no, resizeable=no, scrollbars=no, status=no, titlebar=no,
↳toolbar=no, width=100");
```

А этот сценарий откроет крошечное окошко с размерами 100×100 пикселов, без заголовка (в Internet Explorer версий 4.*x* и 5.*x*), строки меню, панели инструментов, панели адреса, строки статуса, полос прокрутки и без возможности изменить размеры окна.

> ***Народ предупреждает!***
>
> Окна без строки меню, панели инструментов, панели адреса, строки статуса, полос прокрутки и возможности изменить их размеры — очень плохой стиль Web-программирования. Лучше их избегать.

Как получить доступ к созданному программно окну?

Проблема

Мне нужно получить доступ к окну Web-обозревателя, созданного с помощью метода `open` объекта `window`. Как это сделать?

Решение

Воспользоваться значением, которое возвращает метод `open` объекта `window`. Оно представляет собой экземпляр объекта `window`, соответствующий созданному программно окну.

Чтобы получить доступ к содержимому окна, открытого программно, нужно воспользоваться свойством `document` полученного экземпляра объекта `window`. Это свойство предоставляет доступ к соответствующему объекту.

Мы также можем получать доступ к объявленным в новом окне переменным и функциям. Для этого используется такой формат:

```
<Программно созданное окно>.<Переменная> = <Значение>;
<Программно созданное окно>.<Функция>([<Параметры>]);
```

То есть такой же, как и для доступа к свойствам и методам экземпляра объекта.

> ***Народ предупреждает!***
>
> Получить доступ к переменным и функциям, объявленным в созданном программно окне, мы можем только после открытия этого окна и окончания загрузки Web-страницы, которая должна в ней присутствовать.

Пример 1

Приведенный далее сценарий открывает второе окно Web-обозревателя и выводит в нем текст приветствия:

```
<HTML>
  <HEAD>
    <TITLE>Пример</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var secondWindow = window.open("", "second_window");
      secondWindow.document.write("<P>Здравствуйте!</P>");
      secondWindow.document.write("<P>Вас приветствует второе окно
      ↳Web-обозревателя.</P>");
    </SCRIPT>
  </BODY>
</HTML>
```

Пример 2

Напишем код двух Web-страниц jspsOpenWindow1.html и jspsOpenWindow2.htm, из которых первая будет выводить на экран вторую в отдельном окне Web-обозревателя.

Вот код страницы jspsOpenWindow1.html:

```
<HTML>
  <HEAD>
    <TITLE>Первичное окно</TITLE>
    <SCRIPT TYPE="text/javascript">
      var secondWindow = window.open("jspsOpenWindow2.html",
      ↳"second_window");

      function btnOKOnClick()
        {
          secondWindow.greeting = "А я - первичное окно!";
```

```
          secondWindow.writeGreeting();
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM>
      <INPUT TYPE="button" ID="btnOK" VALUE="Нажми!"
      ↳ONCLICK="btnOKOnClick();">
    </FORM>
  </BODY>
</HTML>
```

А вот код страницы jspsOpenWindow2.htm:

```
<HTML>
  <HEAD>
    <TITLE>Вторичное окно</TITLE>
    <SCRIPT TYPE="text/javascript">
      var greeting = "Я - вторичное окно!";

      function writeGreeting()
        {
          document.write("<P>" + greeting + "</P>");
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      writeGreeting();
    </SCRIPT>
  </BODY>
</HTML>
```

Откроем в Web-обозревателе страницу jspsOpenWindow1.html, сдвинем в сторону вторичное окно и в первичном нажмем кнопку **Нажми!**. Текст во вторичном окне изменится.

Как из созданного программно окна получить доступ к создавшему его окну?

В странице, открываемой в созданном программно окне, я пишу Web-сценарий, который должен получить доступ к создающему его окну и выполнять с его содержимым какие-то манипуляции. Как вообще получить к нему доступ?

Решение 1

Использование свойства `opener` объекта `window`. Это свойство возвращает экземпляр объекта `window`, соответствующий окну, из которого было программно создано текущее окно.

Пример

Напишем Web-страницы jspsOpener1.html и jspsOpener2.html, из которых первая будет выводить на экран вторую в отдельном окне Web-обозревателя.

Код страницы jspsOpener1.html таков:

```
<HTML>
  <HEAD>
    <TITLE>Первичное окно</TITLE>
    <SCRIPT TYPE="text/javascript">
      var secondWindow = window.open("jspsOpener2.html",
      ↳"second_window");
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

А так выглядит код страницы jspsOpener2.html:

```
<HTML>
  <HEAD>
    <TITLE>Вторичное окно</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      window.opener.document.write("<P>Вторичное окно открыто!</P>");
    </SCRIPT>
  </BODY>
</HTML>
```

Откроем в Web-обозревателе страницу jspsOpener1.html. После того как откроется вторичное окно, во вторичном окне появится текст "Вторичное окно открыто!".

Решение 2

Можно также создать в программно созданном окне переменную и поместить в нее экземпляр объекта `window`, соответствующий создающему окну. Но это

очень неудобно, поскольку мы можем получить доступ к объявленным в созданном программно окне переменным и функциям только после окончания загрузки страницы в этом окне. Так что лучше этим способом не пользоваться.

Народ замечает

Нужно отметить, что этот способ может быть полезен, если мы имеем сразу несколько программно созданных окон и хотим, чтобы одно из них управляло другим.

Как изменить размеры и местоположение окна?

Проблема

Я хочу установить размеры и местоположение окна Web-обозревателя из сценария. Можно ли это сделать?

Решение

Специально для этого предназначены четыре метода объекта `window`. Все они перечислены далее.

- `moveBy(<Смещение по горизонтали>, <Смещение по вертикали>)` — смещает окно на заданное количество пикселов вправо и вниз. Для перемещения влево и вверх нужно задать отрицательные значения соответствующих параметров.
- `moveTo(<Координата X>, <Координата Y>)` — перемещает окно в точку экрана с заданными координатами.
- `resizeBy(<Приращение по горизонтали>, <Приращение по вертикали>)` — увеличивает размеры окна на заданное количество пикселов. Для уменьшения размеров окна нужно задать отрицательные значения соответствующих параметров.
- `resizeTo(<Размер по горизонтали>, <Размер по вертикали>)` — задает размеры окна.

Народ предупреждает!

Нужно иметь в виду, что многие Web-обозреватели (в частности, Opera и Firefox) позволяют блокировать Web-сценарии, перемещающие окна и изменяющие их размеры.

Пример

Вот HTML-код страницы, автоматически разворачивающей окно Web-обозревателя на весь экран.

```
<HTML>
  <HEAD>
    <TITLE>На весь экран</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      window.moveTo(0, 0);
      window.resizeTo(screen.availWidth - 1, screen.availHeight - 1);
    </SCRIPT>
  </BODY>
</HTML>
```

Народ предупреждает!

Если для определения размеров экрана мы собираемся пользоваться свойствами объекта `screen`, то должны будем иметь в виду, что эти свойства возвращают количество пикселов по горизонтали и вертикали, а нумерация пикселов начинается от 0 и заканчивается величиной *<значение соответствующего свойства>* - 1. Именно поэтому в приведенном выше примере автор уменьшил параметры метода `resizeTo` объекта `window` на единицу.

Как мне получить координаты и размеры окна?

Проблема

Мне нужно получить координаты и размеры окна Web-обозревателя. Это возможно?

Решение

В принципе, возможно, но придется считаться с несовместимостью различных программ Web-обозревателей. Давайте поговорим об этом. И начнем с координат окна Web-обозревателя — их получить проще.

За координаты окна "отвечают" свойства `screenLeft` и `screenTop` объекта `window`. Первое свойство возвращает горизонтальную координату левого верхнего угла окна Web-обозревателя, второе — его вертикальную координату. Эти свойства поддерживаются Internet Explorer и Opera.

Чтобы выяснить координаты окна в Firefox, нам придется обратиться к свойствам `screenX` и `screenY`. Эти свойства также принадлежат объекту `window`.

Перейдем к размерам окна. Здесь лучше всего себя показывают Opera и Firefox — они предоставляют свойства `outerWidth` и `outerHeight` объекта `window`, возвращающие соответственно ширину и высоту окна.

К сожалению, вычислить размеры окна в Internet Explorer не представляется возможным. Можно только узнать размеры клиентской области окна, вос-

пользовавшись свойствами `clientWidth` и `clientHeight` объекта `body`, представляющего тело страницы и вложенного в объект `document`.

Народ замечает

Да, Internet Explorer — замечательная программа, но не без досаднейших огрехов. Ну что стоило разработчикам добавить в его объект `window` какие-нибудь свойства `screenWidth` и `screenHeight`!.. Так ведь не добавили...

Существует, правда, "финт ушами", позволяющий все-таки узнать размеры окна. А именно добавить к значению свойства `clientWidth` 10 пикселов (примерно такой будет толщина рамок окна), а к значению свойства `clientHeight` — 100 пикселов (поправка на толщину рамки окна, его заголовка, главного меню и панелей инструментов). Эти значения выяснены автором книги экспериментально.

Как выровнять окно по краю экрана?

Проблема

Мне нужно выровнять окно Web-обозревателя по определенному краю экрана. Как это сделать?

Решение

Использование универсальной функции `jspsAlignWindow` (листинг 4.1). Формат ее вызова таков:

```
jspsAlignWindow(<Окно>, [<Край экрана>[, <Растягивать окно>]]);
```

Первым параметром передается окно, которое нужно выровнять по краю экрана. Это единственный обязательный параметр функции `jspsAlignWindow`.

Вторым параметром этой функции передается числовое значение, задающее край экрана, по которому нужно выровнять окно. Вот все доступные значения этого параметра (приведены имена псевдоконстант, значения которых можно найти в листинге 4.1):

- `JSPS_AW_LEFT` — по левому краю;
- `JSPS_AW_LEFTTOP` — по левому и верхнему краю экрана (т. е. поместить окно в его левый верхний угол);
- `JSPS_AW_TOP` — по верхнему краю;
- `JSPS_AW_RIGHTTOP` — по правому и верхнему краю (в правый верхний угол);
- `JSPS_AW_RIGHT` — по правому краю;
- `JSPS_AW_RIGHTBOTTOM` — по правому и нижнему краю (в правый нижний угол);
- `JSPS_AW_BOTTOM` — по нижнему краю;

- JSPS_AW_LEFTBOTTOM — по левому и нижнему краю (в левый нижний угол);
- JSPS_AW_CENTER — по центру;
- JSPS_AW_FULL — полное выравнивание (т. е. окно будет растянуто на весь экран).

Если второй параметр не указан, окно будет выровнено по центру, как если бы мы указали значение JSPS_AW_CENTER.

Третий параметр функции jspsAlignWindow позволяет указать, растягивать ли окно вдоль края экрана, по которому выполняется выравнивание, так, чтобы оно заняло его полностью. Если указано значение true, окно будет растянуто; значение false предписывает оставить размеры окна неизменными. Пропуск третьего параметра эквивалентен указанию значения false.

Нужно иметь в виду, что третий параметр функции jspsAlignWindow имеет смысл только при значениях второго параметра, равных JSPS_AW_LEFT, JSPS_AW_TOP, JSPS_AW_RIGHT и JSPS_AW_BOTTOM. В остальных случаях он будет проигнорирован.

Функция jspsAlignWindow не возвращает значения.

Народ предупреждает!

Максимизированные окна функция jspsAlignWindow выравнивает некорректно. И это понятно — если окно развернуто на весь экран, его просто невозможно "прижать" к какому-то одному его краю.

Листинг 4.1. Функция jspsAlignWindow, выполняющая выравнивание окна Web-обозревателя вдоль заданного края экрана

```
var JSPS_AW_LEFT = 0;
var JSPS_AW_LEFTTOP = 1;
var JSPS_AW_TOP = 2;
var JSPS_AW_RIGHTTOP = 3;
var JSPS_AW_RIGHT = 4;
var JSPS_AW_RIGHTBOTTOM = 5;
var JSPS_AW_BOTTOM = 6;
var JSPS_AW_LEFTBOTTOM = 7;
var JSPS_AW_CENTER = 8;
var JSPS_AW_FULL = 9;

function jspsAlignWindow(windowObject, edge, doResize)
  {
    if (typeof(edge) == "undefined") edge = JSPS_AW_CENTER;
    if (!doResize) doResize = false;
```

```
var piObj = jspsGetProgramInfo();
var rightmostPixel = screen.availWidth - 1;
var bottommostPixel = screen.availHeight - 1;
// Здесь выполняется коррекция размеров экрана для Opera с учетом
// того, что окна в этом Web-обозревателе открываются внутри
// главного окна. Значения коррекции установлены автором
// экспериментально
if (piObj.programName == JSPS_GPI_OPERA)
  {
    var correctionX = 10;
    var correctionY = 90;
  }
else
  {
    var correctionX = 0;
    var correctionY = 0;
  }
if ((piObj.programName == JSPS_GPI_MSIE) ||
↳(piObj.programName == JSPS_GPI_OPERA))
  {
    var winX = window.screenLeft;
    var winY = window.screenTop;
  }
else
  {
    var winX = window.screenX;
    var winY = window.screenY;
  }
if (piObj.programName == JSPS_GPI_MSIE)
  {
    var winWidth = window.document.body.clientWidth + 10;
    var winHeight = window.document.body.clientHeight + 100;
  }
else
  {
    var winWidth = window.outerWidth;
    var winHeight = window.outerHeight;
  }
switch (edge)
  {
    case JSPS_AW_LEFT:
      if (doResize)
        {
          windowObject.moveTo(0, 0)
```

```
        windowObject.resizeTo(winWidth, bottommostPixel -
        ↳correctionY);
      }
    else
      windowObject.moveTo(0, winY - correctionY);
    break;
  case JSPS_AW_LEFTTOP:
    windowObject.moveTo(0, 0);
    break;
  case JSPS_AW_TOP:
    if (doResize)
      {
        windowObject.moveTo(0, 0);
        windowObject.resizeTo(rightmostPixel - correctionX,
        ↳winHeight);
      }
    else
      windowObject.moveTo(winX - correctionX, 0);
    break;
  case JSPS_AW_RIGHTTOP:
    windowObject.moveTo(rightmostPixel - winWidth - correctionX,
    ↳0);
    break;
  case JSPS_AW_RIGHT:
    if (doResize)
      {
        windowObject.moveTo(rightmostPixel - winWidth -
        ↳correctionX, 0);
        windowObject.resizeTo(winWidth, bottommostPixel -
        ↳correctionY);
      }
    else
      windowObject.moveTo(rightmostPixel - winWidth - correctionX,
      ↳winY - correctionY);
    break;
  case JSPS_AW_RIGHTBOTTOM:
    windowObject.moveTo(rightmostPixel - winWidth - correctionX,
    ↳bottommostPixel - winHeight - correctionY);
    break;
  case JSPS_AW_BOTTOM:
    if (doResize)
      {
        windowObject.moveTo(0, bottommostPixel - winHeight -
        ↳correctionY);
```

```
            windowObject.resizeTo(rightmostPixel - correctionX,
            ↳winHeight);
          }
        else
          windowObject.moveTo(winX - correctionX, bottommostPixel -
          ↳winHeight - correctionY);
        break;
      case JSPS_AW_LEFTBOTTOM:
        windowObject.moveTo(0, bottommostPixel - winHeight -
        ↳correctionY);
        break;
      case JSPS_AW_CENTER:
        windowObject.moveTo((rightmostPixel - winWidth - correctionX) /
        ↳2, (bottommostPixel - winHeight - correctionY) / 2);
        break;
      case JSPS_AW_FULL:
        windowObject.moveTo(0, 0);
        windowObject.resizeTo(rightmostPixel - correctionX,
        ↳bottommostPixel - correctionY);
        break;
    }
}
```

Внимание!

Листинг 4.1, содержащий объявление функции `jspsAlignWindow`, использует также функцию `jspsGetProgramInfo`, чье объявление приведено в листинге 3.1.

Хорошая идея!

Поместите объявление этой функции в файл сценариев alignwindow.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="alignwindow.js"></SCRIPT>
```

Пример

```
jspsAlignWindow(window, JSPS_AW_BOTTOM, false);
```

Это выражение выровняет текущее окно Web-обозревателя по нижнему краю экрана без растягивания.

Как активизировать нужное окно?

Проблема

Можно ли из сценария принудительно активизировать какое-либо окно Web-обозревателя?

Решение

Использовать не принимающий параметров метод `focus` объекта `window`.

> ***Народ замечает***
>
> Объект `window` также поддерживает не принимающий параметров метод `blur`, деактивирующий окно. Правда, предугадать, какое окно в этом случае будет активизировано, невозможно.

Пример

```
someWindow.focus();
```

Это выражение активизирует программно открытое окно `someWindow`.

```
otherWindow.blur();
```

А это выражение деактивирует программно открытое окно `otherWindow` и делает активным другое окно.

Как узнать положение полос прокрутки?

Проблема

Мне нужно узнать положение полос прокрутки в окне Web-обозревателя. Это возможно?

Решение

Самый удобный способ — это воспользоваться свойствами `scrollLeft` и `scrollTop` объекта `body`, вложенного в объект `document`. Первое свойство возвращает положение горизонтальной полосы прокрутки в пикселах, второе — вертикальной.

> ***Народ замечает***
>
> Opera и Firefox поддерживают также свойства `pageXOffset` и `pageYOffset` объекта `window`. Первое свойство возвращает положение горизонтально полосы прокрутки также в пикселах, второе — вертикальной.

Пример

```
var posHScroll = document.body.scrollLeft;
var posVScroll = document.body.scrollTop;
```

Как выполнить прокрутку содержимого окна?

Проблема

Мне нужно прокрутить открытую в окне Web-обозревателя страницу так, чтобы показать посетителю какой-то ее элемент. Можно ли это сделать, и если можно, то как?

Решение 1

Использовать перечисленные далее методы объекта `window`.

- `scrollBy(<Смещение по горизонтали>, <Смещение по вертикали>)` — прокручивает содержимое окна на заданное количество пикселов вправо и вниз. Для прокрутки влево и вверх нужно задать отрицательные значения соответствующих параметров.
- `scrollTo(<Координата X>, <Координата Y>)` — прокручивает содержимое окна в точку с заданными координатами.

Пример

Вот HTML-код небольшой Web-странички, прокручивающей после открытия свое содержимое на 200 пикселов по вертикали:

```
<HTML>
  <HEAD>
    <TITLE>Прокрутка</TITLE>
  </HEAD>
  <BODY>
    // Разместите здесь какой-либо большой текст в тегах <P>
    <SCRIPT TYPE="text/javascript">
      window.scrollBy(0, 200);
    </SCRIPT>
  </BODY>
</HTML>
```

Решение 2

Все экземпляры объектов, представляющие элементы страницы, поддерживают метод `scrollIntoView`, формат вызова которого таков:

```
<Элемент страницы>.scrollIntoView([<Край окна, у которого должен
↪находиться элемент страницы>]);
```

Единственный необязательный параметр этого метода задает край окна, у которого после прокрутки должен находиться заданный элемент страницы. Если задано значение `true`, элемент страницы появится у верхнего края окна; такой же эффект даст пропуск этого параметра. Значение `false` прокручивает окно так, что элемент страницы будет находиться у нижнего края окна.

> ***Народ предупреждает!***
>
> Этот способ прокрутки содержимого окна не работает в Opera (хотя метод `scrollIntoView`, похоже, там поддерживается). Автору не удалось установить, почему так происходит.

Пример

```
var pObject = document.all["p"];
pObject.scrollIntoView(true);
```

Этот сценарий выполнит прокрутку содержимого окна так, чтобы элемент страницы с именем `p` (оно задается с помощью атрибута `ID`) находился у верхнего края окна.

Решение 3

Присвоить нужно значение в пикселах уже знакомым нам свойствам `scrollLeft` и `scrollTop` объекта `body`, вложенного в объект `document`. Первое свойство "отвечает" за положение горизонтальной полосы прокрутки в пикселах, второе — вертикальной.

Пример

```
document.body.scrollTop = 100;
```

Это выражение вызовет прокрутку содержимого окна Web-обозревателя по вертикали на 100 пикселов.

Как закрыть окно Web-обозревателя?

Проблема

Можно ли из Web-сценария закрыть окно Web-обозревателя? И как это делается?

Решение

Использовать не принимающий параметров метод `close` объекта `window`.

> ***Народ предупреждает!***
>
> Окна Web-обозревателя, открытые программно (методом `open`), при вызове метода `close` будут закрыты без предупреждения. Но при закрытии окон, открытых самим пользователем, на экране появится окно-предупреждение, спрашивающее, действительно ли следует закрыть это окно. Вывод этого предупреждения невозможно запретить ни из Web-сценариев, ни в настройках Web-обозревателя.

Как проверить, было ли созданное программно окно Web-обозревателя закрыто пользователем?

Проблема

Может случиться так, что пользователь закроет окно Web-обозревателя, созданное программно. Можно ли как-то узнать об этом?

Решение

Использовать свойство `closed` объекта `window`. Это свойство вернет значение `true`, если окно закрыто, и `false` в противном случае.

Пример

```
if (!someWindow.closed)
  someWindow.document.write("<P>Глава 14</P>")
```

Этот сценарий выводит в окно `someWindow` абзац с текстом "Глава 14", если это окно не было закрыто.

Как отследить открытие, активизацию, изменение размеров и закрытие окна?

Проблема

Мне требуется отслеживать моменты открытия, активизации, изменения размеров и закрытия окон Web-обозревателя. Как это сделать?

Решение

Использовать события объекта `window`:

- `onBlur` — наступает, когда окно деактивируется;
- `onFocus` — наступает, когда окно активизируется;
- `onLoad` — наступает после окончания загрузки содержимого окна (т. е. Web-страницы);
- `onResize` — наступает при изменении размеров окна;
- `onUnload` — наступает перед выгрузкой Web-страницы в результате перехода на другую страницу или закрытия окна.

Присвоить этим событиям обработчики можно только путем присвоения соответствующей функции нужному свойству экземпляра объекта `window`. Имя этого свойства совпадает с именем события, набранным маленькими буквами.

> ***Народ предупреждает!***
>
> Отследить перемещение окна — увы! — невозможно ни в одной программе Web-обозревателя.

Пример

```
function windowOnLoad()
  {
    window.alert("Страница загружена");
  }
```

```
function windowOnUnload()
  {
    window.alert("Страница выгружена");
  }
window.onload = windowOnLoad;
window.onunload = windowOnUnload;
```

Этот сценарий создает обработчики событий `onLoad` и `onUnload` текущего окна.

Метод `alert` объекта `window` выводит на экран окно-сообщение с текстом, переданным в качестве единственного параметра этого метода, и кнопкой **OK**.

Как перенаправить посетителя на другую Web-страницу?

Проблема

Мне нужно перенаправить посетителя на другую Web-страницу. Есть ли какие-то способы сделать это?

Решение 1

Использовать метатег следующего вида:

```
<META HTTP-EQUIV="Refresh" CONTENT="<Тайм-аут>; URL=<Интернет-адрес>">
```

Этот метатег записывается в заголовке Web-страницы (в теле `<HEAD>`). Встретив его, Web-обозреватель через указанный в секундах *Тайм-аут* откроет Web-страницу с заданным *Интернет-адресом*.

Достоинство этого способа в том, что он работает всегда, даже если исполнение Web-сценариев было отключено пользователем Web-обозревателя. Недостаток — "статичность"; перенаправление выполняется всегда, в любом случае (из-за этого его называют *статичным перенаправлением*).

> ***Народ советует***
>
> Этот способ лучше использовать в тех случаях, когда сайт меняет свой интернет-адрес. В этом случае по старому адресу оставляют страницу, содержащую указанный выше метатег, в параметрах которого записан новый интернет-адрес сайта.

Решение 2

Присвоить нужный интернет-адрес свойству `href` объекта `location`, вложенного в объект `window`.

Этот же способ имеет то неоспоримое достоинство, что позволяет перенаправить посетителя на разные страницы в зависимости от каких-либо условий

(например, используемого им Web-обозревателя). Поэтому такое перенаправление называется *динамическим*.

Пример 1

Вот HTML-код Web-странички, выполняющей перенаправление на другую страницу через 5 секунд после ее открытия:

```
<HTML>
  <HEAD>
    <TITLE>Перенаправление</TITLE>
    <SCRIPT TYPE="text/javascript">
      function bodyOnLoad()
        {
          window.setTimeout(goToOtherPage, 5000);
        }

      function goToOtherPage()
        {
          window.location.href = "OtherPage.html";
        }
    </SCRIPT>
  </HEAD>
  <BODY ONLOAD="bodyOnLoad();">
    <P>Внимание! Через 5 секунд вы будете перенаправлены на другую
    ↳Web-страницу.</P>
  </BODY>
</HTML>
```

Метод `setTimeout` вызывает функцию, переданную в качестве первого параметра, через заданное вторым параметром количество миллисекунд. В нашем случае задано 5000 миллисекунд — 5 секунд.

Пример 2

А вот еще одна Web-страница, перенаправляющая пользователей Internet Explorer на одну страницу, а пользователей других Web-обозревателей — на другую:

```
<HTML>
  <HEAD>
    <TITLE>Перенаправление</TITLE>
    <SCRIPT TYPE="text/javascript">
      function bodyOnLoad()
        {
          var piObject = jspsGetProgramInfo();
```

```
        if (piObject.programInfo == JSPS_GPI_MSIE)
          window.location.href = "index_ie.html"
        else
          window.location.href = "index_others.html"
      }
    </SCRIPT>
  </HEAD>
  <BODY ONLOAD="bodyOnLoad();">
  </BODY>
</HTML>
```

Внимание!

Приведенный пример использует функцию `jspsGetProgramInfo`, чье объявление представлено в листинге 3.1.

Как отключить контекстное меню?

Проблема

Есть ли способ запретить вывод в окне Web-обозревателя контекстного меню при щелчке правой кнопкой мыши?

Решение

Есть, но только в Internet Explorer версии 5.0 или более новой и Firefox. Нужно использовать написать функцию-обработчик события `onContextMenu` объекта `body`, возвращающую значение `false`. (Как мы помним из *главы 2*, возврат из функции-обработчика значения `false` запрещает действие по умолчанию при наступлении данного события.)

Народ предупреждает!

Не следует забывать о трех вещах. Во-первых, Opera не поддерживает событие `onContextMenu`. Во-вторых, Firefox позволяет в своих настройках отключить исполнение сценариев, запрещающих контекстное меню. В-третьих, сценарии, убирающие контекстное меню, — очень плохой стиль Web-программирования; считается, что так ограничивается свобода посетителя сайта пользоваться размещенной на нем информацией, как ему заблагорассудится.

Пример

Далее приведен код HTML Web-страницы, в которой при щелчке правой кнопкой мыши не будет выводиться контекстное меню:

```
<HTML>
  <HEAD>
    <TITLE>Долой контекстное меню!</TITLE>
  </HEAD>
```

```
  <BODY ONCONTEXTMENU="return false;">
    <P>Попробуйте щелкнуть правой кнопкой мыши и проверьте, появится ли
    ↳контекстное меню.</P>
    <P>Ага, не появилось!!!</P>
  </BODY>
</HTML>
```

Здесь мы вставили в значение атрибута `ONCONTEXTMENU` тега `<BODY>`, соответствующего одноименному событию, единственное выражение, возвращающее `false`. В данном случае необязательно писать функцию-обработчик.

Как добавить интернет-адрес открытой страницы в меню *Избранное*?

Проблема

На многих сайтах я видел гиперссылку, при щелчке на которой в меню **Избранное** заносится интернет-адрес этой страницы. Я тоже хочу это сделать на своем сайте!

Решение

Есть способ, но он будет работать только в версиях 4.*x* и 5.*x* Internet Explorer. Для этого нужно использовать выражение, формат которого таков:

```
external.AddFavorite(<Интернет-адрес>[, <Название>]);
```

Объект `external` предоставляет доступ к некоторым возможностям Web-обозревателя, обеспечиваемыми его расширениями, в частности, к меню **Избранное**. Метод `AddFavorite` этого объекта занимается добавлением в список **Избранное** новых пунктов.

Первый параметр этого метода задает интернет-адрес, который нужно добавить в **Избранное**. Необязательный второй параметр задает название этого пункта; если он пропущен, будет использовано название Web-страницы (содержимое тега `<TITLE>`).

> ***Народ советует***
>
> При вызове метода `AddFavorite` объекта `external` для задания значения первого метода лучше всего использовать описанное выше свойство `window.location.href`. Кто знает, может интернет-адрес нашего сайта впоследствии изменится...

При вызове метода `AddFavorite` на экране появится диалоговое окно **Добавление в Избранное** Internet Explorer. Пользователь может либо подтвердить добавление страницы в список **Избранное**, либо отказаться от этого. Метод не возвращает никакого значения.

Народ предупреждает!

Internet Explorer 6.0 SP2 с установленными последними обновлениями уже не дает возможности добавить сайт в **Избранное** таким способом. Вероятно, это сделано для вящей безопасности. Opera и Firefox также не поддерживают эту возможность.

Пример

Небольшая Web-страница, позволяющая добавить себя в **Избранное**:

```
<HTML>
  <HEAD>
    <TITLE>Добавление в Избранное</TITLE>
    <SCRIPT TYPE="text/javascript">
      function hrfOnClick()
        {
          external.AddFavorite(window.location.href);
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P><A HREF="#" ONCLICK="hrfOnClick();">Добавить в избранное</A></P>
  </BODY>
</HTML>
```

Как присвоить сайту значок?

Проблема

Я еще хочу присвоить своему Web-сайту значок, чтобы он отображался в строке адреса правее самого интернет-адреса. Ну очень хочу!

Решение

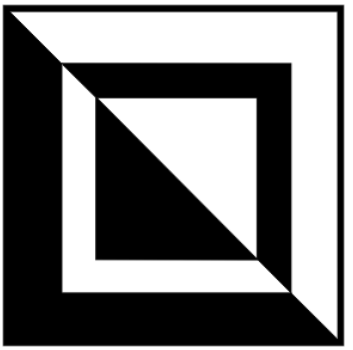
Нужно использовать тег вида:

```
<LINK REL="SHORTCUT ICON" HREF="<Интернет-адрес файла значка>">
```

Этот тег помещается в заголовок Web-страницы (в тег `<HEAD>`).

Что дальше?

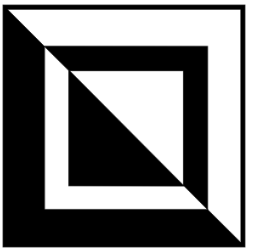
С Web-обозревателем мы закончили. Следующим шагом будут манипуляции содержимым его окон, т. е. с Web-страницами. Начнем мы с простейших эффектов, типа переливающегося текста, мигающих рамок и пр. А уж овладев ими, мы узнаем, как менять содержимое страниц "на лету". И углубимся в дебри объектной модели DOM, так любимой Firefox.



ЧАСТЬ III

Работа с Web-страницами и их элементами

ГЛАВА 5



Простейшие манипуляции и эффекты

В этой главе представлены народные советы о манипуляциях Web-страницами и их элементами. Мы узнаем, как изменить внешний вид элементов страницы и создать простейшие анимационные эффекты (мерцающая рамка вокруг текста, гиперссылка с меняющимся цветом и пр.) и как изменить содержимое страницы после ее загрузки средствами JavaScript.

Управление внешним видом элементов страницы

Начнем с управления внешним видом элементов страницы и создания простейших анимационных эффектов.

Как изменить внешний вид элемента страницы?

Проблема

Мне нужно изменить внешний вид элемента страницы (цвет текста, фона, размеры рамки и др.). Как это делается?

Решение

Все экземпляры объектов, представляющие элементы страницы, содержат вложенный объект `style`. Свойства этого объекта суть атрибуты стиля CSS, привязанного к данному элементу страницы.

Чтобы выяснить имя свойства объекта `style`, соответствующее нужному нам атрибуту стиля, достаточно знать одно несложное правило. Имена этих свойств имеют почти такие же имена, как и атрибуты стиля, только без сим-

волов "тире", а первые буквы всех слов, образующих имя атрибута, кроме первого, делаются большими. В табл. 5.1 показаны примеры преобразования имен атрибутов стиля в имена свойств объекта `style`.

Таблица 5.1. *Примеры преобразования атрибутов стиля в свойства объекта* `style`

Атрибут стиля	Свойство объекта `style`
`background-attachment`	`backgroundAttachment`
`border-bottom-color`	`borderBottomColor`
`font-family`	`fontFamily`
`z-index`	`zIndex`

Присваивая этим свойствам нужные значения, мы можем менять внешний вид элемента страницы. Нужно только помнить, что значения этих свойств должны иметь тот же формат, что и значения соответствующих им атрибутов стиля.

> ***Народ замечает***
>
> Кроме всего этого, экземпляры объектов, представляющих элементы страницы, также поддерживают свойства, соответствующие атрибутам создающих их тегов. Так, экземпляр объекта, соответствующего абзацу текста, поддерживает свойство `align`, задающее выравнивание текста в абзаце и аналогичное атрибуту `ALIGN` тега `<P>`. Но пользоваться этими свойствами не рекомендуется, т. к. они, как и атрибуты форматирования тегов, объявлены устаревшими.

Пример 1

```
var pObject = document.all["p"];
pObject.style.fontSize = "24pt";
```

Этот небольшой сценарий задаст для абзаца с именем `p` размер шрифта, равный 24 пунктам.

Пример 2

А вот HTML-код небольшой Web-странички. При наведении мыши на содержащийся в ней текстовый абзац вокруг него появляется рамка.

```
<HTML>
  <HEAD>
    <TITLE>Текст с рамкой</TITLE>
    <SCRIPT TYPE="text/javascript">
      function pOnMouseOver()
```

```
        {
          var pObject = document.all["p"];
          pObject.style.borderLeftColor = "#000000";
          pObject.style.borderTopColor = "#000000";
          pObject.style.borderRightColor = "#000000";
          pObject.style.borderBottomColor = "#000000";
        }

      function pOnMouseOut()
        {
          var pObject = document.all["p"];
          pObject.style.borderLeftColor = "#FFFFFF";
          pObject.style.borderTopColor = "#FFFFFF";
          pObject.style.borderRightColor = "#FFFFFF";
          pObject.style.borderBottomColor = "#FFFFFF";
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P ID="p" STYLE="border: thin solid #FFFFFF"
    ↳ONMOUSEOVER="pOnMouseOver();" ONMOUSEOUT="pOnMouseOut();">Если вы
    ↳наведете на меня курсор мыши, вокруг меня появится рамка.</P>
  </BODY>
</HTML>
```

Видно, что мы присвоили событиям `onMouseOver` (наведение курсора мыши на элемент страницы) и `onMouseOut` ("увод" курсора мыши с него) функции-обработчики, которые и выполняют задание новых параметров стиля абзаца.

Народ предупреждает!

Мы могли бы написать функции-обработчики событий и так:

```
function pOnMouseOver()
  {
    var pObject = document.all["p"];
    pObject.style.border = "thin solid #000000";
  }

function pOnMouseOut()
  {
    var pObject = document.all["p"];
    pObject.style.border = "thin solid #FFFFFF";
  }
```

но такой код не работал бы в Opera. Не любит этот Web-обозреватель комплексные атрибуты стилей, наподобие `border`, и соответствующие им свойства объекта `style`...

Как временно скрыть элемент страницы?

Проблема

Мне нужно временно скрыть элемент Web-страницы, а потом снова вывести его на экран. Как это сделать?

Решение

Объект `style` поддерживает свойства `display` и `visibility`. Они позволяют временно скрыть элемент страницы, но делают это по-разному.

Свойство `visibility` просто скрывает элемент страницы, делая его невидимым, но при этом не убирает его со страницы. То есть, если мы, скажем, скроем с помощью этого свойства абзац, то он исчезнет со страницы, но вместо него будет отображаться пустое пространство. Нужно иметь это в виду.

Вот все допустимые значения свойства `visibility` объекта `style`:

- `"hidden"` — элемент страницы скрыт;
- `"visible"` — элемент страницы виден;
- `"inherit"` — элемент страницы виден, если виден его родитель, и наоборот.

Что касается свойства `display` объекта `style`, то оно также позволяет скрыть элемент страницы, хотя его назначение несколько иное. Это свойство позволяет задать, как элемент страницы отображается на ней: в виде блочного элемента, в виде встроенного элемента, в виде строки таблицы и пр. Профессиональные Web-дизайнеры говорят, что свойство `display` задает поведение элемента страницы.

Доступных значений этого свойства очень много:

- `"none"` — элемент вообще не будет отображаться на странице, словно он и не задан в ее HTML-коде (это главное отличие свойства `display` от свойства `visibility`);
- `"inline"` — элемент страницы ведет себя как отдельный символ текста *(встроенный элемент)*;
- `"block"` — элемент страницы ведет себя как абзац текста *(блочный элемент)*;
- `"list-item"` — элемент страницы ведет себя как пункт списка;

- "run-in" — *встраивающийся* элемент. Если за таким элементом следует блочный элемент, он становится первым символом блочного элемента, в противном случае он сам становится блочным элементом;
- "compact" — *компактный* элемент. Если за таким элементом следует блочный элемент, он форматируется как однострочный встроенный элемент и помещается левее блочного элемента. В противном случае он сам форматируется как блочный элемент;
- "marker" — маркер списка;
- "table" — таблица;
- "inline-table" — таблица, отформатированная как встроенный элемент;
- "table-row-group" — секция тела таблицы;
- "table-header-group" — секция "шапки" таблицы;
- "table-footer-group" — секция "поддона" таблицы;
- "table-row" — строка таблицы;
- "table-column-group" — определение *группы столбцов* таблицы (формируется с помощью парного тега <COLGROUP> и служит для задания параметров сразу нескольких столбцов);
- "table-column" — определение столбца таблицы (формируется с помощью парного тега <COL> и служит для задания параметров отдельного столбца);
- "table-cell" — ячейка таблицы;
- "table-caption" — заголовок таблицы.

Народ предупреждает!

Internet Explorer поддерживает только значения "none", "inline", "block", "list-item", "table-header-group" и "table-footer-group". Причем поддержка значения "table-footer-group" появилась в версии 5.5, а остальных — в версии 5.

Так вот, свойство display позволяет скрыть элемент страницы полностью, словно он и не был определен в HTML-коде страницы. Для этого достаточно присвоить этому свойству значение "none". Но чтобы вернуть ему видимость, придется вспомнить, чем же был этот элемент страницы до скрытия: встроенным, блочным, таблицей или маркером списка. И присвоить свойству display соответствующее значение.

Народ советует

Если нужно временно скрыть свободно позиционируемый контейнер (о манипуляциях ими будет рассказано в *главе 10*), то можно использовать как свойство visibility, так и свойство display — оба дают один и тот же эффект. Лучше, конечно, использовать свойство visibility — с ним меньше проблем.

Пример

Далее приведен HTML-код Web-страницы с абзацем и кнопкой. При нажатии кнопки абзац пропадет со страницы, при повторном нажатии — снова появится на ней.

```
<HTML>
  <HEAD>
    <TITLE>Скрытие элемента страницы</TITLE>
    <SCRIPT TYPE="text/javascript">
      var flag = true;

      function toggleVisibility()
        {
          var pObject = document.all["p"];
          var btnObject = document.all["btn"];
          if (flag)
            {
              pObject.style.visibility = "hidden";
              btnObject.value = "Показать";
            }
          else
            {
              pObject.style.visibility = "visible";
              btnObject.value = "Скрыть";
            }
          flag = !flag;
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P ID="p">Это содержимое страницы</P>
    <FORM>
      <INPUT TYPE="button" ID="btn" VALUE="Скрыть"
      ↳ONCLICK="toggleVisibility();">
    </FORM>
  </BODY>
</HTML>
```

Чтобы обозначить, скрыт ли в данный момент наш абзац, мы использовали переменную `flag`, хранящую логическое значение. Так будет проще и "дешевле".

Как изменить внешний вид сразу нескольких элементов страницы?

Проблема

А что делать, если мне нужно изменить внешний вид сразу нескольких элементов страницы? Писать код, аналогичный приведенному ранее, для всех этих элементов?

Решение 1 (Internet Explorer и Firefox)

Совсем не обязательно. Можно создать стиль, привязать его ко всем нужным элементам страницы и в сценарии изменить параметры этого стиля.

Чтобы добраться до атрибутов стиля, сначала нужно получить доступ к таблице стилей, в которой он определен. Для этого следует использовать поддерживаемую Internet Explorer и Firefox коллекцию `styleSheets`, вложенную в объект `document`. Элементы этой коллекции соответствуют всем таблицам стилей, встроенным и внешним, определенным в данной Web-странице. Например, выражение

```
var ssFirstObject = document.styleSheets[0];
```

поместит в переменную `ssFirstObject` экземпляр объекта, соответствующий первой определенной в странице таблице стилей.

Добравшись до таблицы стилей, мы можем получить доступ к нужному нам стилю. Для этого мы используем коллекцию `rules` (для Internet Explorer) или `cssRules` (для Firefox), вложенную в объект таблицы стилей. Элементы этих коллекций суть все стили, определенные в данной таблице.

Предположим, что в переменной `ssFirstObject` находится нужная нам таблица стилей. Тогда получить первый определенный в таблице стиль в Internet Explorer мы можем с помощью выражения

```
var stFirstObject = ssFirstObject.rules[0];
```

В Firefox нам придется воспользоваться таким выражением:

```
var stFirstObject = ssFirstObject.cssRules[0];
```

После этого в переменной `stFirstObject` окажется первый стиль этой таблицы.

Получив нужный нам стиль, мы получим доступ к его атрибутам через еще один объект под названием `style`. Например, выражение

```
stFirstObject.style.fontSize = "24pt";
```

задаст для стиля, находящегося в переменной `stFirstObject`, размер шрифта, равный 24 пунктам.

Народ замечает

Ну вот, на этот раз отличилась Opera... В самом деле, что мешало ее разработчикам сделать в ней поддержку коллекции `styleSheets`?..

Пример

```
var piObject = jspsGetProgramInfo();
if (piObject.programName == JSPS_GPI_MSIE)
  document.styleSheets[0].rules[0].style.fontSize = "8pt"
else
  if (piObject.programName == JSPS_GPI_FIREFOX)
    document.styleSheets[0].cssRules[0].style.fontSize = "8pt";
```

Этот сценарий задаст для первого стиля, определенного в первой таблице стилей, размер шрифта, равный 10 пунктам. Причем это произойдет только в Internet Explorer и Firefox.

Внимание!

Приведенный пример использует функцию `jspsGetProgramInfo`, чье объявление приведено в листинге 3.1.

Решение 2

Есть и не столь изящное, но зато универсальное решение, работающее во всех Web-обозревателях. Достаточно перебрать все элементы коллекции `all` объекта `document`, найти нужные и присвоить свойствам, соответствующим атрибутам стиля, новые значения.

Здесь нам помогут следующие свойства экземпляров объектов, представляющих элементы страницы:

- `className` возвращает имя стилевого класса, привязанного к данному элементу страницы;
- `id` возвращает имя элемента, т. е. значение атрибута `ID`;
- `tagName` возвращает имя тега, создающего данный элемент страницы, без символов < и >.

Народ предупреждает!

Значение, возвращенное свойством `tagName`, может быть набрано как маленькими, так и большими буквами, что очень неудобно при сравнении его с эталонным значением. Поэтому нам придется привести его к единообразному виду, воспользовавшись методами `toUpperCase` или `toLowerCase`. Первый метод возвращает строку, набранную только большими буквами, второй — строку, набранную маленькими буквами.

Пример

Вот HTML-код небольшой Web-страницы, в которой после ее открытия шрифт второго и четвертого абзаца увеличивается:

```
<HTML>
  <HEAD>
    <TITLE>Манипуляции стилем</TITLE>
    <STYLE>
      .para { font-size: 10pt; }
    </STYLE>
  </HEAD>
  <BODY>
    <P CLASS="para">Это небольшой текст.</P>
    <P>Это другой небольшой текст.</P>
    <P CLASS="para">Это третий небольшой текст.</P>
    <P>Это четвертый небольшой текст.</P>
    <SCRIPT TYPE="text/javascript">
      for (var i = 0; i < document.all.length; i++)
        {
          var elObject = document.all[i];
          if (elObject.className == "para")
            elObject.style.fontSize = "24pt";
        }
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь мы сначала привязали ко второму и четвертому абзацам стиль `para`. После этого в сценарии выполнили поиск всех элементов страницы, к которым был привязан этот стиль, и задали для них размер шрифта, равный 24 пунктам.

Как реализовать простейшие анимационные эффекты?

Проблема

Я хочу для привлечения внимания сделать мерцающую рамку у одного из абзацев. Как это сделать?

Решение 1

Нет ничего проще! Для этого достаточно менять цвета рамки через определенное время. И в этом нам поможет метод `setInterval` объекта `window`, фор-

мат вызова которого приведен далее. Этот метод занимается тем, что создает так называемый *таймер*, который выполняет заданное нами выражение через определенные интервалы времени.

```
window.setInterval(<Выражение>, <Интервал>);
```

Первым параметром в строковом виде передается выражение JavaScript, которое будет выполняться каждый раз после истечения заданного интервала. Сам интервал в миллисекундах устанавливается вторым параметром.

Выражение, которое мы передаем методу первым параметром, может делать все что угодно. В нашем случае оно может осуществлять изменение цвета рамки абзаца, чтобы создать эффект ее мерцания. Также мы можем с помощью этого выражения обновлять какие-то данные на Web-странице, изменять положение какого-либо из свободно позиционируемых контейнеров (так, кстати, создается "настоящая" анимация) или делать что-то еще.

Народ советует

Анимационные эффекты на Web-странице нужно использовать очень умеренно и только для привлечения внимания к чему-то очень важному. Вряд ли посетителям нашего сайта понравится, если страницы будут мигать на все лады, как новогодняя елка!..

Метод `setInterval` возвращает особый *идентификатор таймера*, который желательно сохранить в переменной. Впоследствии этот идентификатор может нам понадобиться.

Народ предупреждает!

Не следует путать методы `setInterval` и `setTimeout`! Первый выполняет заданное выражение многократно через заданные интервалы времени, в то время как второй делает это всего один раз, после того истечения интервала.

Заданное нами в первом параметре метода `setInterval` выражение будет выполняться до бесконечности, пока мы не закроем Web-страницу или не выполним метод `clearInterval` объекта `window`. Этот метод удаляет созданный методом `setInterval` таймер; формат его вызова таков:

```
window.clearInterval(<Идентификатор таймера>);
```

Первым параметром этому методу передается возвращенный методом `setInterval` идентификатор таймера. Результата метод `clearInterval` не возвращает.

Народ советует

Если мы хотим, чтобы анимационный эффект продолжался бесконечно (в смысле, все время, пока открыта данная страница), идентификатор таймера нет нужды сохранять в переменной.

Пример

Пример Web-страницы с абзацем, имеющим мерцающую рамку:

```
<HTML>
  <HEAD>
    <TITLE>Мерцающая рамка</TITLE>
    <SCRIPT TYPE="text/javascript">
      var isBlinked = false;
      function borderBlink()
        {
          var pObject = document.all["p"];
          if (isBlinked)
            {
              pObject.style.borderLeftColor = "#0000FF";
              pObject.style.borderTopColor = "#0000FF";
              pObject.style.borderRightColor = "#0000FF";
              pObject.style.borderBottomColor = "#0000FF";
            }
          else
            {
              pObject.style.borderLeftColor = "#FF0000";
              pObject.style.borderTopColor = "#FF0000";
              pObject.style.borderRightColor = "#FF0000";
              pObject.style.borderBottomColor = "#FF0000";
            }
          isBlinked = !isBlinked;
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P ID="p" STYLE="border: thin solid #0000FF">У меня мерцающая
    ↳рамка!</P>
    <SCRIPT TYPE="text/javascript">
      window.setInterval("borderBlink()", 2000);
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь, чтобы обозначить состояние, в котором находится мерцающая рамка абзаца, мы также использовали переменную `isBlinked`, хранящую логическое значение. И по возможности будем делать впредь.

Народ предупреждает!

Свойства объекта `style`, задающие цвет, возвращают различные значения в зависимости от программы Web-обозревателя. В случае Internet Explorer и Opera

возвращается значение вида `#RRGGBB`, а Firefox вернет значение `rgb(RR, GG, BB)`, где `RR`, `GG` и `BB` — доля, соответственно, красной, зеленой и синей составляющей в цвете. И здесь пресловутая несовместимость Web-обозревателей!..

Решение 2

Использование универсального объекта `AnimatedElement` (листинг 5.1). Его конструктор имеет следующий формат вызова:

```
AnimatedElement(<Элемент страницы>, <Интервал>, [<Массив выражений>]);
```

Первым параметром в конструктор передается экземпляр объекта, соответствующий элементу страницы, к которому мы хотим применить анимационный эффект. Второй параметр задает интервал времени в секундах (секунды нам все-таки привычнее). Тут все нам уже знакомо еще по методу `setInterval` объекта `window`.

А теперь — внимание! Третий параметр конструктора представляет собой массив строк, каждая из которых содержит выражение JavaScript. Все эти выражения будут выполняться после истечения *Интервала*: сначала — выражение, содержащееся в первой строке-элементе массива, потом — во второй строке, далее — в третьей, в четвертой и, наконец, в последней; затем снова будет выполнено выражение в первой строке, и круг замкнется. Таких выражений в массиве может быть задано сколько угодно, но присутствовать должны хотя бы два.

Объект `AnimatedElement` имеет два метода, которые можно использовать в сценариях:

- `start` — запуск анимационного эффекта;
- `stop` — остановка анимационного эффекта.

Оба метода не принимают параметров и не возвращают результата.

Остальные методы и свойства этого объекта использовать не рекомендуется, т. к. объект может повести себя непредсказуемо. Если же понадобится изменить анимационный эффект, лучше всего уничтожить экземпляр объекта, создающего этот эффект, и создать его заново, с другими параметрами.

Народ предупреждает!

Перед уничтожением экземпляра объекта `AnimatedElement` следует вызвать метод `stop` этого объекта.

Листинг 5.1. Объект `AnimatedElement`, создающий простейший анимационный эффект для элемента страницы

```
// Счетчик созданных к данному времени экземпляров объекта
// AnimatedElement
var jsps_AE_counter = 0;
```

```
// Список всех созданных к данному времени экземпляров объекта
// AnimatedElement
var jsps_AE_list = new Array();

// Функция-конструктор
function AnimatedElement(pElement, pInterval, pSt)
  {
    this.element = pElement;
    this.interval = pInterval * 1000;
    this.actions = new Array();
    for (var i = 0; i < pSt.length; i++)
      this.actions[i] = pSt[i];
    this.start = mjspsAEStart;
    this.stop = mjspsAEStop;
    this.timer = null;
    this.actionNumber = 0;
    // Номер данного экземпляра объекта в списке jsps_AE_list
    this.myNumber = jsps_AE_counter;
    // Заносим данный экземпляр объекта в список jsps_AE_list
    jsps_AE_list[jsps_AE_counter] = this;
    // Увеличиваем счетчик экземпляров этого объекта на единицу,
    // подготавливая его для создания нового экземпляра
    jsps_AE_counter++;
    this.start();
  }

// Функция, реализующая метод start. Заметим, что мы передаем в функцию
// jspsAEDoAction, выполняющую очередное из заданных выражений, номер
// данного экземпляра объекта AnimatedElement. Это нужно, чтобы данная
// функция смогла получить к нему доступ
function mjspsAEStart()
  {
    this.timer = window.setInterval("jspsAEDoAction(" +
    ↳this.myNumber.toString() + ")", this.interval);
  }

// Функция, реализующая метод stop
function mjspsAEStop()
  {
    if (this.timer)
      window.clearInterval(this.timer);
    this.timer = null;
  }
```

```
// Функция, выполняющая очередное из заданных выражений. Для доступа
// к данному экземпляру объекта AnimatedElement она использует
// список jsps_AE_list и передаваемый ей единственным параметром номер
// экземпляра этого объекта
function jspsAEDoAction(myNumber)
  {
    var my = jsps_AE_list[myNumber];
    eval(my.actions[my.actionNumber]);
    if (my.actionNumber >= my.actions.length - 1)
      my.actionNumber = 0
    else
      my.actionNumber++;
  }
```

Хорошая идея!

Поместите объявление этого объекта в файл сценариев animatedelement.js. Впоследствии, чтобы использовать его, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="animatedelement.js"></SCRIPT>
```

Пример

Переделанный пример Web-страницы с абзацем, имеющим мерцающую рамку:

```
<HTML>
  <HEAD>
    <TITLE>Анимационный эффект</TITLE>
    <SCRIPT SRC="animatedelement.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта AnimatedElement было
    помещено в файл сценариев animatedelement.js -->
    <SCRIPT TYPE="text/javascript">
      function btnOnClick()
        {
          var btnTriggerObject = document.all["btnTrigger"];
          if (btnTriggerObject.value == "Мерцать!")
            {
              anObject.start();
              btnTriggerObject.value = "Не мерцать!";
            }
          else
            {
              anObject.stop();
              btnTriggerObject.value = "Мерцать!";
            }
        }
```

```
    </SCRIPT>
  </HEAD>
  <BODY>
    <P ID="p" STYLE="border: thin solid #0000FF">У меня мерцающая
    ↳рамка!</P>
    <FORM>
      <INPUT TYPE="button" ID="btnTrigger" VALUE="Не мерцать!"
      ↳ONCLICK="btnOnClick();">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      var pObject = document.all["p"];

      function doMyAction(colorString)
        {
          pObject.style.borderLeftColor = colorString;
          pObject.style.borderTopColor = colorString;
          pObject.style.borderRightColor = colorString;
          pObject.style.borderBottomColor = colorString;
        }

      var anObject = new AnimatedElement(pObject, 2,
      ↳new Array("doMyAction('#FF0000');", "doMyAction('#00FF00');",
      ↳"doMyAction('#0000FF');"));
    </SCRIPT>
  </BODY>
</HTML>
```

Как изменить содержимое страницы после ее загрузки?

Проблема

Тяжела доля Web-программиста!.. Мне нужно изменить содержимое Web-страницы после ее загрузки, а я не знаю, как это сделать.

Решение 1

Самое простое — это воспользоваться методом `write` объекта `document`. Этот не возвращающий значения метод выводит переданную в качестве единственного параметра строку на Web-страницу. При этом строка будет выведена в том месте, где встретился вызов метода `write`, и все теги HTML, встретившиеся в этой строке, будут обработаны.

Народ советует

Это самый простой и самый "совместимый" способ дополнить содержимое страницы после ее загрузки. Но для изменения ее содержимого он — увы! — не подходит.

Пример

Далее приведен HTML-код страницы, выводящей после загрузки текущую дату.

```
<HTML>
  <HEAD>
    <TITLE>Простейший вывод</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var d = new Date();
      document.write("<P>Сегодняшняя дата - " + d.toLocaleString() +
      ↳"</P>");
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь выражение, содержащее метод `write` объекта `document`, выводит на страницу абзац (тег `<P>`), содержащий текстовое представление даты.

Решение 2 (Internet Explorer и Opera)

Internet Explorer версий 4.0 и выше и Opera идут навстречу Web-программистам, которым нужно менять содержимое страниц уже после того, как они загрузятся. Далее перечислены свойства объекта, представляющего элемент страницы, которые можно для этого использовать:

- `innerHTML` — предоставляет доступ к содержимому элемента страницы, которое рассматривается как код HTML;
- `innerText` — предоставляет доступ к содержимому элемента страницы, которое рассматривается как обычный текст;
- `outerHTML` — предоставляет доступ к содержимому элемента страницы, которое рассматривается как код HTML, а также к тегам, формирующим сам этот элемент;
- `outerText` — предоставляет доступ к текстовому содержимому элемента страницы, а также к тегам, формирующим сам этот элемент.

Давайте рассмотрим эти свойства подробнее. И сделаем это на примере.

Предположим, у нас имеется абзац, HTML-код которого таков:

```
<P ID="para">Это просто <EM>курсивный</EM> текст.</P>
```

Воспользуемся свойством `innerText`, чтобы получить его текстовое содержимое:

```
var paraObject = document.all["para"];
var str = paraObject.innerText;
```

В переменной `str` окажется строка "Это просто курсивный текст.", без всяких тегов HTML. То есть при обращении к свойству `innerText` все HTML-теги из возвращаемого им значения исключаются.

Теперь давайте присвоим этому свойству новое значение:

```
paraObject.innerText = "А это <STRONG>другой</STRONG> текст.";
```

В результате в окне Web-обозревателя мы увидим вот это:

```
А это <STRONG>другой</STRONG> текст.
```

Обратим внимание, что теги HTML не были обработаны, а просто оказались выведенными на экран, словно это обычный текст.

А теперь опробуем в действии свойство `innerHTML`:

```
var paraObject = document.all["para"];
paraObject.innerHTML = "А это <STRONG>другой</STRONG> текст.";
```

И получим на экране вот что:

А это **другой** текст.

То есть все теги HTML были правильно обработаны, и слово "другой" было выделено полужирным шрифтом.

Свойство `outerHTML` позволит нам заменить не только внутреннее содержимое элемента страницы, но и сам этот элемент, в смысле, теги, которые его формируют. Давайте возьмем описанный выше абзац и присвоим его свойству `outerHTML` другой HTML-код:

```
var paraObject = document.all["para"];
paraObject.outerHTML = "<DIV>А это <STRONG>другой</STRONG> текст.</DIV>";
```

Получим элемент страницы, HTML-код которого будет таким:

```
<DIV>А это <STRONG>другой</STRONG> текст.</DIV>
```

То есть мы фактически создали новый элемент страницы — не абзац (тег `<P>`), а контейнер (тег `<DIV>`).

Свойство `outerText` в этом смысле самое радикальное. Оно позволяет заменить новым текстом и содержимое элемента страницы, и сам этот элемент, причем все теги HTML, и форматирующие содержимое этого элемента, и

создающие его, будут исключены. То есть, если мы присвоим свойству `outerText` нашего абзаца новый текст:

```
var paraObject = document.all["para"];
paraObject.outerText = "А это другой текст.";
```

то получим простой текст

```
А это другой текст.
```

который будет принадлежать внешнему по отношению к абзацу элементу страницы (в нашем случае — телу страницы).

> ***Народ советует***
>
> Свойства `outerHTML` и `outerText` могут быть полезны, если нужно быстро удалить какой-либо элемент страницы. В этом случае достаточно присвоить любому из этих свойств пустую строку:
>
> ```
> var paraObject = document.all["para"];
> paraObject.outerText = "";
> ```

Пример

```
<H1 ID="mainHeader">Здравствуйте!</H1>
<SCRIPT TYPE="text/javascript">
  var mainHeaderObject = document.all["mainHeader"];
  mainHeaderObject.innerText = "До свидания!";
</SCRIPT>
```

Этот сценарий заменит заголовок "Здравствуйте!" на "До свидания!".

```
<H1 ID="mainHeader">Здравствуйте!</H1>
<SCRIPT TYPE="text/javascript">
  var mainHeaderObject = document.all["mainHeader"];
  mainHeaderObject.innerHTML = "<EM>До свидания!</EM>";
</SCRIPT>
```

А этот сценарий еще и выделит его курсивом.

> ***Народ предупреждает!***
>
> Напоминаем, что все описанные здесь свойства поддерживаются только Internet Explorer и Opera. Firefox, что называется, отдыхает.

Решение 3

Использование методов и свойств DOM. Этот способ много сложнее, чем описанный выше, но работает во всех Web-обозревателях.

Полное описание DOM выходит за рамки настоящей книги. Автор просто приведет список методов и свойств, относящихся к DOM.

Народ советует

Желающие узнать все о DOM могут наведаться на соответствующий раздел сайта W3C, расположенный по адресу **http://www.w3.org/DOM/DOMTR**.

Метод `createElement` объекта `document` позволяет создать элемент страницы, являющийся тегом, одинарным или парным. Формат его вызова таков:

```
document.createElement(<Имя тега>);
```

Имя тега передается в строковом виде и без символов < и >. Метод возвращает экземпляр объекта, соответствующий созданному элементу страницы.

Например, выражение

```
var pObject = document.createElement("P");
```

создаст обычный текстовый абзац и поместит соответствующий ему экземпляр объекта в переменную `pObject`.

Народ предупреждает!

Парные теги, создаваемые таким образом, не имеют содержимого.

Метод `createTextNode` того же объекта позволяет создать фрагмент текста, который может потом стать содержимым парного тега. Формат его вызова таков:

```
document.createTextNode(<Текст>);
```

Разумеется, *Текст* передается в строковом виде. Метод возвращает экземпляр объекта, соответствующий созданному текстовому фрагменту.

Так, выражение

```
var textObject = document.createTextNode("Это содержимое тега.");
```

создаст фрагмент текста "Это содержимое тега." и поместит соответствующий ему экземпляр объекта в переменную `textObject`.

Но созданный таким образом элемент страницы не будет сразу же помещен на Web-страницу. Нам придется сделать это самим, для чего мы воспользуемся методом `appendChild` объекта, соответствующего элементу страницы, в который мы хотим вложить вновь созданный элемент (в терминологии DOM он называется *родителем*). Вот его формат вызова:

```
<Родитель>.appendChild(<Добавляемый элемент страницы>);
```

После вызова этого метода *Добавляемый элемент страницы* будет вложен в *Родителя* и, как еще говорят, станет его *потомком*.

```
pObject.appendChild(textObject);
document.body.appendChild(pObject);
```

Первое из приведенных выше выражений поместит созданный ранее текстовый фрагмент `textObject` в абзац `pObject`, сделав его содержимым этого абза-

ца. Второе же выражение поместит абзац `pObject` в тело Web-страницы, после чего он и появится в окне Web-обозревателя.

Метод `appendChild` помещает *Добавляемый элемент страницы* в конец *Родителя*. Если же мы хотим поместить его в начало или середину *Родителя*, то воспользуемся методом `insertBefore`:

```
<Родитель>.insertBefore(<Добавляемый элемент страницы>,
↳<Потомок, перед которым будет помещен добавляемый элемент>);
```

Думается, тут особо нечего комментировать...

Теперь временно отвлечемся от методов и рассмотрим различные полезные свойства DOM объекта, соответствующего элементу страницы. Они перечислены в табл. 5.2.

Таблица 5.2. *Полезные свойства DOM объекта, соответствующего элементу страницы*

Свойство	Описание
`childNodes`	Возвращает коллекцию потомков данного элемента страницы
`firstChild`	Возвращает экземпляр объекта, соответствующий первому потомку данного элемента страницы. Если он не имеет потомков, возвращается `null`
`lastChild`	Возвращает экземпляр объекта, соответствующий последнему потомку данного элемента страницы. Если он не имеет потомков, возвращается `null`
`nextSibling`	Возвращает экземпляр объекта, соответствующий следующему потомку того же родителя, потомком которого является данный элемент страницы. Если таковых больше нет, возвращается `null`
`nodeValue`	Для текстового фрагмента возвращает его содержимое. Для тега возвращает `null`
`parentNode`	Возвращает экземпляр объекта, соответствующий родителю. Если родителя нет или если данный элемент страницы был создан, но еще не помещен на Web-страницу, возвращается `null`
`previousSibling`	Возвращает экземпляр объекта, соответствующий предыдущему потомку того же родителя, потомком которого является данный элемент страницы. Если таковых нет, возвращается `null`

Поясним назначение этих свойств.

Ну, со свойствами `firstChild`, `lastChild` и `childNodes` все понятно. Они позволяют получить доступ к первому потомку, последнему потомку и всем сразу потомкам данного элемента страницы соответственно.

```
var lastPObject = document.body.lastChild;
```

Это выражение поместит в переменную `lastPObject` экземпляр объекта, соответствующий последнему добавленному в Web-страницу элементу. Это будет абзац, с которым мы имели дело ранее.

```
var secondChildObject = document.body.childNodes[1];
```

Это выражение поместит в переменную `secondChildObject` экземпляр объекта, соответствующий второму по счету элементу Web-страницы.

```
document.body.insertBefore(pObject, document.body.childNodes[0]);
```

А это выражение поместит созданный нами ранее абзац `pObject` в самое начало страницы (перед первым потомком).

Свойства `nextSibling` и `previousSibling` позволяют получить, соответственно, следующий и предыдущий элементы страницы того же уровня вложенности (имеющие того же родителя, что и текущий элемент). А свойство `parentNode` возвращает экземпляр объекта, соответствующий родителю текущего элемента. Здесь также все просто.

Свойство `nodeValue` — самый быстрый способ получить доступ к содержимому фрагмента текста. Вот так мы можем задать новое содержимое для недавно созданного нами абзаца:

```
document.body.lastChild.firstChild.nodeValue =
↳"Это новое содержимое тега.";
```

Заметим, что здесь мы сначала получили доступ к самому абзацу с помощью строки свойств (она называется *путем*) `document.body.lastChild`, а потом добрались до его содержимого — текстового фрагмента — через свойство `firstChild`. Да, зачастую пути доступа к нужному объекту, свойству или методу DOM бывают очень длинными...

Вернемся к методам объекта, соответствующего элементу страницы. Метод `replaceChild` позволяет заменить любой потомок данного элемента:

```
<Родитель>.replaceChild(<Заменяющий элемент страницы>,
↳<Заменяемый потомок>);
```

Этот метод возвращает экземпляр объекта, соответствующий *Заменяемому потомку*.

Метод `removeChild` удаляет потомок из элемента страницы:

```
<Родитель>.removeChild(<Удаляемый потомок>);
```

и возвращает экземпляр объекта, соответствующий *Удаляемому потомку*.

Не принимающий параметров метод `hasChildNodes` возвращает `true`, если элемент страницы имеет потомков, и `false` в противном случае.

Очень полезный метод `cloneNode` позволяет получить точную копию элемента страницы. Вызывается он так:

```
<Элемент страницы>.cloneNode(<Включать потомки и атрибуты>);
```

Единственный параметр этого метода должен быть логической величиной. Если он равен `true`, в копию элемента страницы будут включены все его потомки и атрибуты, если они были созданы. Значение `false` предписывает выполнить копирование только самого этого элемента страницы. Метод возвращает экземпляр объекта, соответствующий копии *Элемента страницы*.

> ***Народ советует***
>
> Если нам понадобится создать очень много однотипных элементов страниц (например, ячеек и строк таблицы), метод `cloneNode` будет исключительно полезен.

А что же атрибуты тегов? За атрибуты "отвечает" целый набор методов, которые мы сейчас рассмотрим.

Метод `setAttribute` объекта, соответствующего элементу страницы, позволяет добавить новый атрибут в тег, который этот элемент формирует, или изменить его значение, если такой атрибут уже присутствует. Вот формат вызова этого метода:

```
<Элемент страницы>.setAttribute(<Имя атрибута>, <Значение атрибута>);
```

Оба параметра этого метода передаются в строковом виде.

Например:

```
document.body.lastChild.setAttribute("id", "para");
```

Это выражение задает для нашего абзаца имя `para`.

> ***Народ предупреждает!***
>
> С помощью этого метода можно создавать обработчики событий. Для этого достаточно указать в параметрах имя атрибута, соответствующее нужному событию, и код его обработчика:
>
> ```
> document.body.lastChild.setAttribute("onclick", "someFunction();");
> ```
>
> Но созданные таким образом обработчики события будут поддерживаться только Opera и Firefox. Если же мы хотим задавать таким образом обработчики событий, которые бы поддерживались Internet Explorer, нам придется слегка нарушить формат вызова метода `setAttribute`, передав вторым параметром не строку, содержащую выражение, а саму функцию-обработчик:
>
> ```
> document.body.lastChild.setAttribute("onclick", someFunction);
> ```
>
> Но заданные таким образом обработчики событий будут поддерживаться, наоборот, только Internet Explorer. Поэтому лучше не задавать обработчики событий методом `setAttribute`, а использовать для этого способы, описанные в *главе 2*.

Метод `getAttribute` позволяет получить значение атрибута:

```
<Элемент страницы>.getAttribute(<Имя атрибута>);
```

Имя атрибута передается в строковом виде. Метод возвращает строку, содержащую значение атрибута. Если значение не задано, возвращается значение по умолчанию или пустая строка, если для данного атрибута значение по умолчанию не определено.

Метод `hasAttribute` позволяет проверить, определен ли данный атрибут в теге, который формирует данный элемент страницы:

```
<Элемент страницы>.hasAttribute(<Имя атрибута>);
```

Имя атрибута передается в строковом виде. Метод возвращает `true`, если такой атрибут определен, и `false` в противном случае.

А метод `removeAttribute` удаляет атрибут из тега:

```
<Элемент страницы>.removeAttribute(<Имя атрибута>);
```

И здесь *Имя атрибута* передается в строковом виде. Метод не возвращает значения.

> ***Народ предупреждает!***
>
> Internet Explorer отличился и здесь — он не "находит" атрибуты, имена которых заданы большими буквами. Поэтому всегда следует задавать эти имена маленькими буквами.

Кроме того, существует не принимающий параметров метод `hasAttributes`, возвращающий `true`, если тег, который формирует элемент страницы, содержит атрибуты со значениями, отличными от значений по умолчанию, и `false` в противном случае.

Пример

Далее приведен код HTML Web-страницы, в которой программно создается текстовый абзац вида

Щелкните, *пожалуйста*, по мне!

где слово "пожалуйста" выделено курсивом. При щелчке мышью по этому абзацу будет выведено окно-сообщение.

```
<HTML>
  <HEAD>
    <TITLE>Демонстрация работы DOM</TITLE>
    <SCRIPT TYPE="text/javascript">
      // Функция-обработчик события onClick абзаца
      function paraOnClick()
```

```
        {
          window.alert("Вы щелкнули по абзацу");
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      // Создаем сам абзац
      var pObject = document.createElement("P");
      // Задаем имя абзаца
      pObject.setAttribute("id", "para");
      // Создаем первую часть текста абзаца
      var text1Object = document.createTextNode("Щелкните, ");
      // Создаем элемент-потомок — тег <EM>, в который будет заключена
      // вторая часть текста абзаца
      var emObject = document.createElement("EM");
      // Создаем вторую часть текста абзаца
      var text2Object = document.createTextNode("пожалуйста");
      // Помещаем вторую часть текста абзаца в тег <EM>
      emObject.appendChild(text2Object);
      // Создаем третью часть текста абзаца
      var text3Object = document.createTextNode(", по мне!");
      // Помещаем все эти части текста в абзац
      pObject.appendChild(text1Object);
      pObject.appendChild(emObject);
      pObject.appendChild(text3Object);
      // Помещаем абзац в тело Web-страницы
      document.body.appendChild(pObject);
      // Создаем обработчик события onClick абзаца
      document.all["para"].onclick = paraOnClick;
    </SCRIPT>
  </BODY>
</HTML>
```

Народ советует

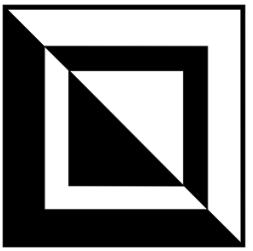
Вообще, самый простой способ привязать к событию создаваемого программно абзаца функцию-обработчик — это сделать вот так:

```
pObject.onclick = paraOnClick;
```

Как говорится, дешево и сердито.

Что дальше?

Закончив со скучным текстом, мы вскоре перейдем к работе с графикой. И произойдет это в следующей главе.



ГЛАВА 6

Работа с графикой

В этой главе представлены народные советы по работе с графическими изображениями. Мы научимся подменять одно изображение другим, создавать *"горячее"* изображение, меняющееся при наведении курсора мыши, создавать простейшую анимацию путем быстрой смены изображений и даже рисовать на Web-странице произвольную фигуру. Вот как!

Простейшие эффекты с изображениями

Простейшие эффекты с изображениями — это подмена изображения, "горячее" изображение и анимация. Этим мы займемся в первую очередь.

Как заменить одно изображение другим после загрузки страницы?

Проблема

Мне нужно заменить находящееся на Web-странице изображение другим уже после загрузки этой страницы. Как это сделать?

Решение

Объект, соответствующий графическому изображению, поддерживает свойство `src`, задающее интернет-адрес файла изображения. Чтобы сменить изображение, достаточно присвоить этому свойству интернет-адрес нового файла.

Пример

```
<IMG ID="img1" SRC="picture1.jpg">
. . .
```

```
<SCRIPT TYPE="text/javascript">
  var img1Object = document.images["img1"];
  img1Object.src = "picture2.jpg";
</SCRIPT>
```

Этот сценарий подменяет изображение, выведенное на Web-страницу из файла picture1.jpg, на изображение из файла picture2.jpg.

Народ советует

Для получения доступа к изображениям на странице удобнее использовать коллекцию `images` объекта `document` (как это сделал автор книги в приведенном выше примере). Эта коллекция обычно значительно меньше, чем "глобальная" коллекция `all`, поэтому поиск в ней выполняется быстрее.

Как создать изображение, меняющееся при наведении на него курсора мыши?

Проблема

Я сделал гиперссылку в виде графического изображения (изображение-гиперссылку) и теперь хочу, чтобы при наведении на него курсора мыши оно менялось. Что мне нужно для этого сделать?

Решение 1

Да всего лишь привязать к событиям `onMouseOver` (наступает при наведении курсора мыши на изображение) и `onMouseOut` (наступает при "уводе" курсора мыши с изображения) тега `<IMG>`, формирующего это изображение, функции-обработчики, которые будут менять одно изображение на другое. И "горячее" изображение готово!

Народ предупреждает!

Нужно проследить, чтобы заменяющее изображение имело те же геометрические размеры, что и изначальное. Иначе получится очень некрасиво.

Пример

Вот фрагмент кода страницы, который формирует "горячее" изображение.

```
<A HREF="#">
  <IMG ID="img1" SRC="picture1.jpg"
  ↳ONMOUSEOVER="img1Object.src = 'picture2.jpg';"
  ↳ONMOUSEOUT="img1Object.src = 'picture1.jpg';">
</A>
```

```
<SCRIPT TYPE="text/javascript">
  var img1Object = document.images["img1"];
</SCRIPT>
```

Как видим, обработчики событий `onMouseOver` и `onMouseOut` в нашем случае получились очень простыми — в одно выражение. Они используют для доступа к экземпляру объекта, соответствующему изображению, переменную `img1Object`, которая объявляется в расположенном после тега `<IMG>` сценарии. Эта переменная будет готова к использованию сразу же после загрузки и вывода на экран содержимого тега `<IMG>` и окончания обработки следующего за ним сценария.

Решение 2

Использование универсального объекта `HotImage` (листинг 6.1). Его конструктор имеет следующий формат вызова:

```
HotImage(<Изначальное изображение>, <Заменяющее изображение>
↳[, <Интернет-адрес>[, <Цель>]]);
```

Первые два параметра задают интернет-адреса изображений — изначального и заменяющего. Они передаются в строковом виде.

Третий, необязательный, параметр задает также в строковом виде интернет-адрес, по которому произойдет переход при щелчке на "горячем" изображении. Если он не задан, создается "горячее" изображение, не являющееся гиперссылкой.

Четвертый, также необязательный, параметр задает цель "горячего" изображения, т. е. куда будет выведена страница, интернет-адрес которой передан третьим параметром. Если четвертый параметр пропущен, новая страница выводится в то же окно Web-обозревателя, что и текущая (поведение гиперссылок по умолчанию). Цель также задается в строковом виде.

Вот возможные значения четвертого параметра функции-конструктора объекта `HotImage`:

- "`_blank`" — загрузка страницы в новом окне Web-обозревателя;
- "`_parent`" и "`_top`" — загрузка страницы во всем окне Web-обозревателя;
- "`_self`" — загрузка страницы в том же самом окне или фрейме (тот же самый эффект будет, если четвертый параметр пропущен);
- "*<Имя окна или фрейма>*" — загрузка страницы в окне или фрейме с заданным именем.

Единственный метод `putHere` объекта `HotImage` помещает HTML-код, создающий "горячее" изображение, в то место, где находится выражение с его вызовом. Этот метод не принимает параметров и не возвращает результата.

Листинг 6.1. Объект `HotImage`, создающий на Web-странице "горячее" изображение

```
// Функция-конструктор
function HotImage(pImageSrc, pAltImageSrc, pHref, pTarget)
  {
    this.imageSrc = pImageSrc;
    this.altImageSrc = pAltImageSrc;
    this.href = pHref;
    this.target = pTarget;
    this.putHere = mjspsPutHere;
  }

// Функция, реализующая метод putHere
function mjspsPutHere()
  {
    var s = "<IMG SRC=\"" + this.imageSrc +
    ↳"\" ONMOUSEOVER=\"this.src='" + this.altImageSrc +
    ↳"';\" ONMOUSEOUT=\"this.src='" + this.imageSrc + "';\"";
    var sh = "";
    var sh2 = "";
    if (this.href)
      {
        var sh = "<A HREF=\"" + this.href + "\"";
        if (this.target)
          sh += " TARGET=\"" + this.target + "\"";
        sh += ">";
        s += " BORDER=\"0\"";
        sh2 = "</A>";
      }
    s += ">";
    s = sh + s + sh2;
    document.write(s);
  }
```

Хорошая идея!

Поместите объявление этого объекта в файл сценариев hotimage.js. Впоследствии, чтобы использовать его, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="hotimage.js"></SCRIPT>
```

Пример

```
<SCRIPT TYPE="text/javascript">
  var hiObject = new HotImage("picture1.jpg", "picture2.jpg",
  ↳"jspsHotImage.html", "_blank");
  hiObject.putHere();
</SCRIPT>
```

Этот сценарий создает на Web-странице "горячее" изображение. Не правда ли, он короче, чем приведенный ранее пример?

Как создать простейшую анимацию из графических изображений?

Проблема

У меня есть несколько изображений, и я хочу создать на их основе простейшую анимацию. Как это сделать?

Решение

Ну, тут все просто: одно изображение очень быстро сменяет другое. Реализуется это с помощью приема, описанного ранее, и таймера, создаваемого с помощью метода `setInterval` объекта `window`, который мы рассмотрели в *главе 5*.

Пример

Далее приведен фрагмент HTML-кода Web-страницы, на которой реализована подобная простейшая анимация.

```
<IMG ID="img1" SRC="picture1.jpg">
<SCRIPT TYPE="text/javascript">
  var img1Object = document.images["img1"];
  var imagesArray = new Array("picture1.jpg", "picture2.jpg",
  ↳"picture3.jpg", "picture4.jpg");
  var imageIndex = 0;
  function changeImage()
    {
      img1Object.src = imagesArray[imageIndex];
      if (imageIndex >= imagesArray.length - 1)
        imageIndex = 0
      else
        imageIndex++;
    }
  var timer = window.setInterval("changeImage();", 1000);
</SCRIPT>
```

Для хранения интернет-адресов файлов, содержащих изображения-"кадры" нашего "фильма", мы создали массив `imagesArray`. Далее мы просто последовательно выбираем из этого массива один интернет-адрес за другим и загружаем в элемент страницы `img1` соответствующие им изображения.

Народ замечает

Вообще, это не очень удачный способ создания анимации. Лучше использовать фильмы в формате "анимированный GIF" или Shockwave/Flash — для их воспроизведения не придется писать сценарий.

Как вывести на Web-страницу случайно выбранное изображение?

Проблема

Я собираюсь сделать на своем сайте баннерную систему. Как мне вывести на страницу произвольный баннер?

Решение

Здесь тоже все просто. Нужен только массив, содержащий интернет-адреса всех файлов, хранящих баннеры, и сценарий, вычисляющий псевдослучайное число, которое станет индексом элемента этого массива.

Пример

Далее приведен фрагмент HTML-кода страницы, выводящей на экран произвольное изображение.

```
<IMG ID="img1">
<SCRIPT TYPE="text/javascript">
  var imagesArray = new Array("picture1.jpg", "picture2.jpg",
  ↳"picture3.jpg", "picture4.jpg");
  var imageIndex = Math.round(jspsGetRandom(0, imagesArray.length - 1));
  var img1Object = document.images["img1"];
  img1Object.src = imagesArray[imageIndex];
</SCRIPT>
```

Внимание!

Приведенный пример использует функцию `jspsGetRandom`, чье объявление представлено в листинге 1.18.

Народ советует

Поместите код, создающий массив с интернет-адресами файлов с баннерами, в отдельный файл сценариев. Так мы сможем изменить набор баннеров в нашей баннерной системе, не затрагивая код страниц, на которых они должны выводиться.

Как загрузить все нужные изображения до загрузки содержимого страницы?

Проблема

Я по вашему совету создал на своей странице "горячие" изображения. Теперь, когда посетитель наводит на такое изображение курсор мыши, на его месте появляется пустая рамка, а само нужное изображение выводится только через несколько секунд! Что мне делать?

Объяснение

Когда Web-сценарий выводит на страницу заменяющее изображение, Web-обозреватель начинает его загружать из Интернета, что требует некоторого времени. Все это время посетитель созерцает на месте этого изображения пустую рамку...

Решение этой проблемы очень простое — нужно каким-то образом заставить Web-обозреватель загрузить это изображение во время загрузки тела страницы или вообще до этого. (Опытные Web-программисты говорят в таких случаях, что требуется выполнить *предзагрузку* изображений.) Загруженные таким образом изображения Web-обозреватель сохранит в своем кэше и потом будет извлекать именно из него, что много быстрее, чем загрузка их из Интернета.

Решение 1

Поместить перед "горячими" изображениями теги `<IMG>`, задачей которых будет загрузка всех заменяющих изображений. Это самое простое и очевидное решение, и в этом его, пожалуй, единственное достоинство.

Недостатков у такого подхода три. Во-первых, нам придется задать ширину и высоту таких изображений равными одному пикселу, чтобы они не присутствовали на странице. Делается это с помощью атрибутов, соответственно, `WIDTH` и `HEIGHT` тега `<IMG>`. Во-вторых, даже если мы и зададим размеры этих изображений равными одному пикселу, они все равно будут присутствовать на странице, а значит, находиться в памяти компьютера, занимая ее попусту. В-третьих, если мы решим дополнить или изменить набор "горячих" изображений на нашей странице, нам придется переделывать очень много HTML-кода, "отвечающего" за предзагрузку изображений.

Решение 2

Написать сценарий, который будет заниматься предзагрузкой, и поместить его в секции заголовка Web-страницы (в теге `<HEAD>`). Этот сценарий будет выполнять задачи, перечисленные далее.

1. Создаст массив.
2. Создаст экземпляр объекта `Image` (этот объект представляет графическое изображение).
3. Присвоит свойству `src` этого экземпляра объекта интернет-адрес очередного изображения, которое нужно загрузить до загрузки страницы.
4. Присвоит созданный на втором шаге экземпляр объекта `Image` очередному элементу массива, который был создан на первом шаге.
5. Если это не последнее предзагружаемое изображение, перейдет ко второму шагу.

Этот подход не имеет недостатков предыдущего. Предзагружаемые изображения не присутствуют на странице и, следовательно, не занимают памяти компьютера. А чтобы дополнить или изменить набор предзагружаемых изображений, нам будет достаточно исправить выполняющий предзагрузку сценарий, что заметно проще.

Народ советует

В дальнейшем лучше использовать именно этот подход. Собственно, все Web-дизайнеры его и используют.

Пример

Вот пример сценария, выполняющего предзагрузку изображений, чьи интернет-адреса хранятся в элементах массива `imagesArray`.

```
var imagesArray = new Array("picture1.jpg", "picture2.jpg",
↳"picture3.jpg", "picture4.jpg");
var preloadArray = new Array();
for (var i = 0; i < imagesArray.length - 1; i++)
  {
    preloadArray[i] = new Image;
    preloadArray[i].src = imagesArray[i];
  }
```

Решение 3

Использование универсальной функции `jspsPreloadImages` (листинг 6.2), формат вызова которой таков:

```
jspsPreloadImages(<Массив интернет-адресов файлов, где хранятся
↳изображения>);
```

Интернет-адреса должны помещаться в массив в строковом виде. Функция не возвращает значения.

Листинг 6.2. Функция `jspsPreloadImages`, выполняющая предзагрузку изображений

```
function jspsPreloadImages(pImagesArray)
  {
    var preloadArray = new Array();
    for (var i = 0; i < pImagesArray.length - 1; i++)
      {
        preloadArray[i] = new Image;
        preloadArray[i].src = pImagesArray[i];
      }
  }
```

Хорошая идея!

Поместите объявление этой функции в файл сценариев preloadimages.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="preloadimages.js"></SCRIPT>
```

Пример

Переделанный вариант приведенного выше сценария, выполняющего предзагрузку файлов изображений, чьи интернет-адреса хранятся в массиве `imagesArray`:

```
var imagesArray = new Array("picture1.jpg", "picture2.jpg",
"picture3.jpg", "picture4.jpg");
jspsPreloadImages(imagesArray);
```

Как рисовать на Web-странице?

Проблема

Мне очень нужно нарисовать кое-что прямо на Web-странице. Использовать для этого соответствующие приложения Shockwave/Flash, апплеты Java и элементы ActiveX я не могу. Я даже готов смириться с некоторыми ограничениями, связанными с рисованием на Web-странице (наверняка эти ограничения будут присутствовать, поскольку Web-страница — все-таки текстовый документ). Как это можно сделать?

Решение

Да, Web-страница — это текстовый, а не графический документ, поэтому рисовать на ней просто так не получится. (Хотя вывести текст проще простого, в чем мы и убедились еще в *главе 5*.) Но существует описанный в [2] прием,

который позволит нам рисовать прямо на Web-странице, не пользуясь никакими сторонними средствами.

1. Готовим графическое изображение, представляющее собой точку размером 1×1 пиксел того цвета, которым мы хотим рисовать.
2. Создаем новый стиль CSS, задающий абсолютное позиционирование (значение "absolute" атрибута стиля position) и, возможно, другие параметры.
3. В сценарии, реализующем рисование, вычисляем координаты очередной точки фигуры, которую мы рисуем.
4. В том же сценарии создаем новое графическое изображение, выводящее на экран созданную нами на первом шаге точку, задаем ему вычисленные на предыдущем шаге координаты и привязываем к нему стиль, созданный на шаге 2.
5. Если это не последняя точка рисуемой фигуры, переходим к шагу 3.

> ***Народ замечает***
>
> Не самый изящный способ рисования... Скорее, будет полезен как иллюстрация, что рисование на Web-странице в принципе возможно, и как учебный пример. Похоже, именно поэтому автор и не стал особо развивать эту тему.
>
> С другой стороны, часто ли вам приходится рисовать прямо на Web-странице?..

Вот несколько полезных функций, рисующих различные фигуры. Все они были разработаны автором книги на основе кода, представленного в [2].

Функция jspsDrawDot (листинг 6.3) рисует точку с заданными координатами внутри заданного элемента страницы. Ее формат вызова:

```
jspsDrawDot(<Координата X>, <Координата Y>[, <Размер точки>
↳[, <Элемент страницы>]]);
```

Если *Размер точки* не указан, он составит 1 пиксел. Если не указан *Элемент страницы*, точка будет нарисована прямо внутри тела Web-страницы.

Для задания интернет-адреса файла, содержащего само изображение точки, функция jspsDrawDot использует переменную jsps_DD_dotFileName. Эта переменная объявляется в листинге 6.3 перед объявлением данной функции и изначально хранит "универсальное" значение "/dot.gif" (интернет-адрес файла dot.gif, находящегося в корневой папке сайта). Если кому-то из Web-программистов, использующих эту функцию, потребуется другое "рисующее" изображение, он просто присвоит интернет-адрес хранящего его файла этой переменной.

> ***Народ советует***
>
> Кстати, хорошая идея — хранить используемые во многих функциях и нечасто изменяемые параметры в глобальных переменных. Только нужно подобрать подходящие значения этих переменных, которые подошли бы в большинстве случаев.

Функция `jspsDrawDot` не возвращает значения.

Листинг 6.3. Функция `jspsDrawDot`, рисующая на Web-странице точку

```
// Объявляем переменную jsps_DD_dotFileName, хранящую интернет-адрес
// файла, содержащего "рисующую" точку
var jsps_DD_dotFileName = "/dot.gif";

function jspsDrawDot(x, y, pSize, element)
  {
    if (!pSize) pSize = 1;
    if (!element) element = document.body;
    var imgObject = document.createElement("IMG");
    imgObject.style.position = "absolute";
    imgObject.style.width = pSize.toString();
    imgObject.style.height = pSize.toString();
    imgObject.style.left = x.toString();
    imgObject.style.top = y.toString();
    imgObject.src = jsps_DD_dotFileName;
    element.appendChild(imgObject);
  }
```

Функция `jspsDrawLine` (листинг 6.4) рисует прямую линию. Ее формат вызова:

```
jspsDrawLine(<Координата X начала>, <Координата Y начала>,
↳<Координата X конца>, <Координата Y конца>[, <Размер точки>
↳[, <Элемент страницы>]]);
```

Назначение параметров этой функции понятно без объяснений. Она использует объявленную ранее функцию `jspsDrawDot` для рисования точек, из которых будет состоять рисуемая линия, и также не возвращает значения.

Листинг 6.4. Функция `jspsDrawLine`, рисующая на Web-странице прямую линию

```
function jspsDrawLine(x1, y1, x2, y2, pSize, element)
  {
    // Страхуемся на тот случай, если координаты заданы числами
    // с плавающей точкой
    x1 = Math.floor(x1);
    x2 = Math.floor(x2);
    y1 = Math.floor(y1);
    y2 = Math.floor(y2);
    if (!pSize) pSize = 1;
    if (!element) element = document.body;
```

```
  if (y1 == y2)
    {
      // "Быстрое" рисование горизонтальной линии
      var imgObject = document.createElement("IMG");
      imgObject.style.position = "absolute";
      imgObject.style.width = Math.abs(x2 - x1).toString();
      imgObject.style.height = pSize.toString();
      imgObject.style.left = ((x2 > x1) ? x1 : x2).toString();
      imgObject.style.top = y1.toString();
      imgObject.src = jsps_DD_dotFileName;
      element.appendChild(imgObject);
    }
  else
    if (x1 == x2)
      {
        // "Быстрое" рисование вертикальной линии
        var imgObject = document.createElement("IMG");
        imgObject.style.position = "absolute";
        imgObject.style.width = pSize.toString();
        imgObject.style.height = Math.abs(y2 - y1).toString();
        imgObject.style.left = x1.toString();
        imgObject.style.top = ((y2 > y1) ? y1 : y2).toString();
        imgObject.src = jsps_DD_dotFileName;
        element.appendChild(imgObject);
      }
    else
      {
        // Рисование наклонных линий
        var k = Math.abs((y2 - y1) / (x2 - x1));
        if (y1 > y2) k = -k;
        var x = x1;
        while (x != x2)
          {
            jspsDrawDot(x, y1 + k * Math.abs(x - x1), pSize, element);
            if (x2 > x1)
              x += .5
            else
              x -= .5;
            // Мы увеличиваем значение горизонтальной координаты на
            // 0,5 пиксела. Это сделает наши линии более, если так
            // можно сказать, монолитными
          }
      }
}
```

Народ замечает

Интересную штуку придумал автор — он рисует горизонтальные и вертикальные линии, просто растягивая по горизонтали или вертикали рисующую точку. Это и быстрее, чем вырисовывание таких линий поточечно, и компьютерных ресурсов для отображения отнимает много меньше.

Функция `jspsDrawRectangle` (листинг 6.5) занимается рисованием прямоугольников. Формат ее вызова таков:

```
jspsDrawLine(<Координата X верхнего левого угла>,
↳<Координата Y верхнего левого угла>,
↳<Координата X нижнего правого угла>,
↳<Координата Y нижнего правого угла>
↳[, <Размер точки>[, <Элемент страницы>]]);
```

Она использует объявленную ранее функцию `jspsDrawLine` и не возвращает значения.

Листинг 6.5. Функция `jspsDrawRectangle`, рисующая на Web-странице прямоугольник

```
function jspsDrawRectangle(x1, y1, x2, y2, pSize, element)
  {
    jspsDrawLine(x1, y1, x2, y1, pSize, element);
    jspsDrawLine(x2, y1, x2, y2, pSize, element);
    jspsDrawLine(x2, y2, x1, y2, pSize, element);
    jspsDrawLine(x1, y2, x1, y1, pSize, element);
  }
```

Функция `jspsDrawEllipsis` (листинг 6.6) рисует на Web-странице эллипс (или, как частный случай эллипса, окружность, в зависимости от заданных параметров). Вот формат ее вызова:

```
jspsDrawEllipsis(<Координата X центра>, <Координата Y центра>,
↳<Горизонтальный радиус>[, <Вертикальный радиус>[, <Размер точки>
↳[, <Элемент страницы>]]]);
```

Все параметры этой функции должны быть понятны без всяких объяснений. Единственное — если *Вертикальный радиус* не задан, его значение принимается равным *Горизонтальному радиусу* (в результате будет нарисована окружность — вырожденный вариант эллипса).

Функция `jspsDrawEllipsis` использует объявленную ранее функцию `jspsDrawDot` для рисования точек, и также не возвращает значения.

Листинг 6.6. Функция `jspsDrawEllipsis`, рисующая на Web-странице эллипс

```
function jspsDrawEllipsis(x, y, Rh, Rv, pSize, element)
  {
    if (!Rv) Rv = Rh;
    if (!pSize) pSize = 1;
    if (!element) element = document.body;
    var rr = Math.PI * 2;
    for (var a = 0; a <= rr; a += .02)
      jspsDrawDot(x + Rh * Math.sin(a), y + Rv * Math.cos(a), pSize,
      ↳element);
  }
```

Хорошая идея!

Поместите объявление всех функций, выполняющих рисование, и переменной `jsps_DD_dotFileName` в файл сценариев draw.js. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="draw.js"></SCRIPT>
```

Пример 1

Вот HTML-код Web-страницы, выводящей изображение окружности, вписанного в нее квадрата и вписанного уже в этот квадрат треугольника.

```
<HTML>
  <HEAD>
    <TITLE>Рисование на Web-странице</TITLE>
    <SCRIPT SRC="draw.js"></SCRIPT>
    <!-- Предполагаем, что объявления всех использованных в этом коде
    функций находятся в файле сценариев draw.js -->
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      // Задаем имя файла, хранящего "рисующую" точку. Если мы оставим
      // старое значение, страница не будет работать правильно без
      // Web-сервера
      jsps_DD_dotFileName = "dot.gif";
      // Рисуем окружность
      jspsDrawEllipsis(200, 200, 100, 100);
      // Рисуем вписанный в нее квадрат
      var h = Math.sin(Math.PI / 4) * 100;
      jspsDrawRectangle(200 - h, 200 - h, 200 + h, 200 + h);
```

```
      // Рисуем вписанный в квадрат треугольник
      jspsDrawLine(200, 200 - h, 200 - h, 200 + h);
      jspsDrawLine(200, 200 - h, 200 + h, 200 + h);
    </SCRIPT>
  </BODY>
</HTML>
```

Пример 2

Этот небольшой сценарий выведет на Web-страницу синусоиду.

```
var V = 100;
var amp = 50;
var end = 360;
for (var a = 0; a <= end; a++)
  jspsDrawDot(a, V - amp * Math.sin(jspsToRad(a)));
```

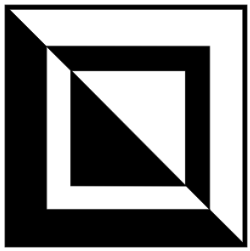
Здесь в переменной `V` хранится отступ от верха страницы до горизонтальной координатной оси, в переменной `amp` — амплитуда ("размах") нашей синусоиды, а в переменной `end` — граничное значение.

> ***Народ предупреждает!***
>
> Не забываем о том, что пикселы на экране отсчитываются слева направо, но сверху вниз! Именно поэтому в выражении, вычисляющем вертикальную координату точки, — `V - amp * Math.sin(jspsToRad(a))` — стоит знак вычитания.

> ***Внимание!***
>
> Приведенный выше пример использует функцию `jspsToRad`, чье объявление представлено в листинге 1.19.

Что дальше?

Ну вот, с графическими изображениями и связанными с ними эффектами мы при помощи всезнающего народа разобрались. На очереди — гиперссылки и все, что с ними связано. Мы научимся выполнять Web-сценарии в ответ на щелчки по ним, создавать всплывающие подсказки и полосы навигации. За дело!



ГЛАВА 7

Работа с гиперссылками и средствами навигации

Настала пора народу пойти войной на гиперссылки и средства навигации по Web-сайту — "горячие" изображения и полосы навигации. (Собственно, гиперссылки — это тоже средства навигации, но самые простые.) Мы же пристроимся в арьергарде народного воинства, посмотрим на военные действия издали и поучимся великой науке побеждать.

Как выполнить Web-сценарий в ответ на щелчок по гиперссылке?

Проблема

Мне нужно, чтобы в ответ на щелчок по гиперссылке выполнялся написанный мной Web-сценарий. Как это сделать?

Решение 1

Самое простое и очевидное решение — оформить нужный сценарий в виде функции-обработчика и привязать его к событию `onClick` гиперссылки.

Пример

Вот HTML-код страницы, содержащей гиперссылку, в ответ на щелчок по которой выполняется сценарий.

```
<HTML>
  <HEAD>
    <TITLE>Гиперссылка с сюрпризом</TITLE>
  </HEAD>
  <BODY>
```

```
    <P><A HREF="#" ONCLICK="window.alert('Вы щелкнули на
    ↳гиперссылке.');">Щелкните на мне!</A></P>
  </BODY>
</HTML>
```

Здесь мы поместили обработчик события `onClick` гиперссылки прямо в значение соответствующего атрибута.

Народ советует

Сам того не ведая, автор решил проблему обновления содержимого сразу нескольких фреймов при щелчке на одной-единственной гиперссылке. Нужно просто написать функцию-обработчик, которая будет выполнять загрузку других Web-страниц в нужные фреймы, и привязать ее к событию `onClick` гиперссылки. Мы еще поговорим об этом в *главе 9*.

Решение 2

Еще более простое, но менее очевидное решение — поместить сценарий, который должен быть выполнен при щелчке на гиперссылке, прямо в значение атрибута `HREF` тега `<A>`. При этом значение должно быть такого вида:

```
javascript:<Сценарий>
```

Обратим внимание, что перед кодом сценария в таком случае должно присутствовать слово "javascript:" (двоеточие обязательно, пробелы между ним и кодом сценария не допускаются).

Пример

Переделанный вариант Web-страницы, HTML-код которой был приведен ранее:

```
<HTML>
  <HEAD>
    <TITLE>Гиперссылка с сюрпризом</TITLE>
  </HEAD>
  <BODY>
    <P><A HREF="javascript:window.alert('Вы щелкнули на
    ↳гиперссылке.');">Щелкните на мне!</A></P>
  </BODY>
</HTML>
```

Народ замечает

Пожалуй, этот вариант используется чаще всего, поскольку он компактнее.

Как заменить интернет-адрес и цель гиперссылки?

Проблема

Мне нужно из сценария заменить интернет-адрес и цель гиперссылки. Как это сделать?

Решение

Это тоже очень просто. Объект, соответствующий гиперссылке, поддерживает свойства `href` и `target`, соответствующие интернет-адресу и цели. Значения обоих свойств задаются в строковом виде.

Вот все возможные значения цели гиперссылки:

- "`_blank`" — загрузка страницы в новом окне Web-обозревателя;
- "`_parent`" и "`_top`" — загрузка страницы во всем окне Web-обозревателя;
- "`_self`" — загрузка страницы в том же самом окне или фрейме (тот же самый эффект будет, если четвертый параметр пропущен);
- "*<Имя окна или фрейма>*" — загрузка страницы в окне или фрейме с заданным именем.

Пример

Далее приведен HTML-код страницы с гиперссылкой, интернет-адрес которой меняется с одного на другой после щелчка по ней.

```
<HTML>
  <HEAD>
    <TITLE>Гиперссылка с секретом</TITLE>
    <SCRIPT TYPE="text/javascript">
      // Объявляем переменную-индикатор, какая страница должна быть
      // открыта после щелчка на гиперссылке
      var flag = false;

      function hrf1OnClick()
        {
          var hrf1Object = document.all["hrf1"];
          hrf1Object.href = (flag) ? "page1.html" : "page2.html";
          flag = !flag;
        }
    </SCRIPT>
  </HEAD>
```

```
  <BODY>
    <P><A ID="hrf1" HREF="" ONCLICK="hrf1OnClick();"
    ↳TARGET="_blank">Угадай, какая страница!</A></P>
  </BODY>
</HTML>
```

Здесь нужно заметить три очень важные вещи. Рассмотрим их по порядку.

Во-первых, мы не указали интернет-адреса гиперссылки в теге `<A>`, который ее создает. Поскольку мы все равно подставляем в гиперссылку интернет-адрес программно, указывать его в теге не обязательно (меньше возни, да и HTML-код страницы станет несколько компактнее).

Во-вторых, мы открываем страницы page1.html и page2.html в новом окне Web-обозревателя (это достигается присвоением значения `"_blank"` атрибуту `TARGET` тега `<A>`). Это обязательно, иначе наш пример не будет работать. Давайте выясним, почему.

Предположим, что мы не включили в тег `<A>` атрибут `TARGET`, и теперь страницы page1.html и page2.html загружаются в том же окне Web-обозревателя, что и тестовая страница. Загрузив тестовую страницу, щелкнем на гиперссылке. Откроется страница page2.htm. При этом тестовая страница будет выгружена, и все объявленные в ней переменные, в том числе и переменная `flag`, будут удалены из памяти компьютера. Если мы теперь вернемся на тестовую страницу, нажав кнопку **Назад** на панели инструментов Web-обозревателя, то переменная `flag` будет заново инициализирована и получит значение `false`. И при щелчке на гиперссылке мы снова попадем на страницу page2.html.

Web-обозреватели не предоставляют стандартных путей сохранить значение какой-либо переменной в памяти после выгрузки страницы, где она была инициализирована. (Есть, правда, один подходящий в данном случае способ — использование cookie; о нем будет рассказано в *главе 13*.) Так что приходится выкручиваться и таким образом...

И, в-третьих, заметим, что сам переход на новую страницу (page1.html или page2.html — не важно) происходит уже после выполнения обработчика события `onClick`. Этим можно пользоваться для подмены интернет-адреса или цели гиперссылки после щелчка на ней (чем мы, собственно, и занимались).

Как создать "горячее" изображение?

Проблема

Мне нужно создать "горячее" изображение, т. е. изображение-гиперссылку, которое при наведении на него курсора мыши меняет свой вид. Как это сделать?

Решение

Создание "горячих" изображений было подробно описано в *главе 6*. Там же был приведен исходный код объявления универсального объекта `HotImage` (см. листинг 6.1), создающего в нужном месте Web-страницы "горячее" изображение при минимальном программировании.

Как создать всплывающие подсказки для гиперссылок?

Проблема

Я видел на одном сайте всплывающие подсказки, появляющиеся на экране при наведении курсора мыши на гиперссылку и выводящие поясняющий текст. Как они делаются?

Решение 1

Самый простой способ реализовать всплывающие подсказки для гиперссылок и вообще любых элементов страницы — это поместить текст, который должен выводиться в подсказках, в атрибуты `TITLE` тегов, с помощью которых создаются элементы страницы.

```
<<Тег> TITLE=<Текст подсказки> [<Другие атрибуты тега>]>
```

Также все объекты, соответствующие элементам страницы, поддерживают свойство `title`, аналогичное атрибуту `TITLE`. Это свойство можно использовать для создания всплывающих подсказок программно.

В тегах `<IMG>`, `<AREA>` и `<INPUT TYPE="image">` для тех же целей можно использовать атрибут `ALT`

```
<IMG ALT=<Текст подсказки> [<Другие атрибуты тега>]>
```

и аналогичное ему свойство `alt`.

Пример

```
<P><A HREF="page1.html" TITLE="На страницу 1">Страница 1</A></P>
<P><A HREF="page2.html" TITLE="На страницу 2">Страница 2</A></P>
<P><A HREF="page3.html" TITLE="На последнюю страницу">Страница 3</A></P>
```

Этот HTML-код создает три гиперссылки, при наведении на которые курсора мыши на экране появятся всплывающие подсказки с текстом, заданным в атрибутах `TITLE` тегов `<A>`.

Решение 2

Несколько более трудоемкое. Нужно написать две функции — обработчики событий `onMouseOver` и `onMouseOut`. Мы помним, что эти события возникают, соответственно, при наведении курсора мыши на элемент страницы (в нашем случае — на гиперссылку) и при "уводе" его прочь.

Функция-обработчик события `onMouseOver` будет выполнять следующие задачи:

- создаст на Web-странице свободно позиционируемый контейнер, который и станет всплывающей подсказкой, и сделает его скрытым (для этого достаточно присвоить свойству `visibility` объекта `style` значение `"hidden"`);
- задаст ему подходящие координаты (они, например, могут совпадать с координатами курсора мыши на момент наведения его на элемент страницы, или же вычисляться на основе координат этого элемента);
- поместит внутрь него нужный текст;
- сделает его видимым, присвоив свойству `visibility` объекта `style` значение `"visible"`.

Все, всплывающая подсказка создана!

На долю функции-обработчика события `onMouseOut` выпадет задача много проще — удалить со страницы созданный первой функцией свободно позиционируемый контейнер. И всплывающая подсказка пропадет.

Здесь возможны варианты. Так, свободно позиционируемый контейнер, который и станет всплывающей подсказкой, можно создавать отдельно, специальным сценарием, помещенным в заголовок страницы (в тег `<HEAD>`). А описанные выше функции — обработчики событий `onMouseOver` и `onMouseOut` будут только делать его видимым и скрывать. Такие подсказки будут выводиться на страницу быстрее, поскольку не придется каждый раз создавать их "с нуля".

> ***На заметку***
>
> На сайте института (**http://life.vgi.volsu.ru**), в котором работает автор, также реализованы всплывающие подсказки. Но только там они перемещаются вслед за курсором мыши. На взгляд автора, это неудобно — ну кому понравится, если текст подсказки елозит вслед за мышью!.. Его и прочитать толком не удастся...

Можно вообще не использовать свободно позиционируемые контейнеры, а выводить текст подсказок в какой-либо абзац или ячейку таблицы. Но, согласитесь, "плавающие" над страницей "окошки" с текстом выглядят и нагляднее, и эффектнее. Особенно для нас, привыкших к красивым всплывающим подсказкам Windows.

Пример

Далее приведен HTML-код простой Web-страницы, содержащей все те же три гиперссылки, при наведении на которые курсора мыши ниже будут появляться подсказки.

```
<HTML>
  <HEAD>
    <TITLE>Пример</TITLE>
    <SCRIPT TYPE="text/javascript">
      function hrfOnMouseOver(pText)
        {
          var outputObject = document.all["output"];
          var textObject = document.createTextNode("")
          textObject.nodeValue = pText;
          outputObject.appendChild(textObject);
        }

      function hrfOnMouseOut()
        {
          var outputObject = document.all["output"];
          outputObject.removeChild(outputObject.firstChild);
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P><A HREF="#" ONMOUSEOVER="hrfOnMouseOver('На страницу 1');"
    ↳ONMOUSEOUT="hrfOnMouseOut();">Страница 1</A></P>
    <P><A HREF="#" ONMOUSEOVER="hrfOnMouseOver('На страницу 2');"
    ↳ONMOUSEOUT="hrfOnMouseOut();">Страница 2</A></P>
    <P><A HREF="#" ONMOUSEOVER="hrfOnMouseOver('На последнюю страницу');"
    ↳ONMOUSEOUT="hrfOnMouseOut();">Страница 3</A></P>
    <P ID="output"></P>
  </BODY>
</HTML>
```

Здесь для вывода текста подсказок мы использовали обычный абзац `output`. Изначально он пуст, поэтому для вывода текста нам придется создать фрагмент текста и поместить его в этот абзац в качестве потомка. Все это делается знакомыми нам по *главе 5* методами `createTextNode` и `appendChild`. Чтобы скрыть подсказку после "увода" курсора мыши с гиперссылки, нам достаточно удалить этот текстовый фрагмент, для чего мы воспользуемся также знакомым нам методом `removeChild`.

Решение 3

Воспользоваться универсальным объектом `PopupTip` (листинг 7.1), реализующим вывод настоящих всплывающих подсказок для любых элементов страницы. Его функция-конструктор имеет очень простой формат вызова:

```
PopupTip([<Стилевой класс для подсказки>]);
```

Единственным, да и то необязательным, параметром в конструктор передается имя стилевого класса, который будет привязан к свободно позиционируемому контейнеру, реализующему всплывающую подсказку. Это имя передается в строковом виде. Если же оно не указано, конструктор задаст для создаваемого контейнера параметры по умолчанию (в этом случае подсказка будет иметь тот же вид, что и системные всплывающие подсказки Windows).

Что касается стилевого класса, который мы привязываем к свободно позиционируемому контейнеру — всплывающей подсказке, то мы можем использовать в нем любые атрибуты и любые их значения, задавая внешний вид подсказки в довольно широких пределах. В частности, мы можем задать цвет текста и фона, шрифт, выравнивание, рамку, отступы и границы. Единственное: нам не следует задавать в нем метод позиционирования (атрибут стиля `position`) и координаты (атрибуты стиля `left` и `top`), поскольку это сделает сам объект `PopupTip`.

Для привязки к элементу страницы всплывающей подсказки используется метод `setTip` этого объекта. Вот формат его вызова:

```
setTip(<Элемент страницы>[, <Текст всплывающей подсказки>]);
```

Если второй параметр не указан, в качестве текста подсказки будет использовано значение свойства `title`, которое поддерживается всеми объектами — элементами страницы и предоставляет доступ к текстовому описанию данного элемента. Это описание задается атрибутом `TITLE`, поддерживаемым практически всеми тегами. Собственно, об этом уже говорилось ранее.

Если к данному элементу страницы уже была привязана всплывающая подсказка, метод `setTip` заменяет старый ее текст новым, переданным ему вторым параметром.

Метод `removeTip` объекта `PopupTip` позволяет удалить всплывающую подсказку, привязанную ранее к элементу страницы. Формат его вызова очень прост:

```
removeTip(<Элемент страницы>);
```

Оба этих метода значения не возвращают.

Свойство `popupTipStyle` объекта `PopupTip` предоставляет доступ к стилю свободно позиционируемого контейнера — всплывающей подсказки. С помощью этого свойства мы можем изменить внешний вид всплывающей подсказки

программно. Но, опять же, не следует менять метод позиционирования (свойство `position`) и координаты (атрибуты `left` и `top`).

Свойство `enabled` этого объекта позволит временно запретить вывод всплывающих подсказок, для чего достаточно присвоить этому свойству значение `false`. Чтобы разрешить вывод подсказок, нужно присвоить этому свойству значение `true`.

Листинг 7.1. Объект `PopupTip`, реализующий всплывающие подсказки для элементов страниц

```
// Счетчик созданных к данному времени экземпляров объекта PopupTip
var jsps_PT_counter = 0;
// Список всех созданных к данному времени экземпляров объекта PopupTip
var jsps_PT_list = new Array();

// Функция-конструктор
function PopupTip(pClassName)
  {
    // Создаем свободно позиционируемый контейнер, который и будет
    // реализовывать всплывающие подсказки, и задаем его стиль
    var popupElement = document.createElement("DIV");
    if (pClassName)
      popupElement.className = pClassName
    else
      {
        popupElement.style.backgroundColor = "#FFFFCC";
        popupElement.style.borderLeftStyle = "solid";
        popupElement.style.borderLeftColor = "#CCCCCC";
        popupElement.style.borderLeftWidth = "1";
        popupElement.style.borderTopStyle = "solid";
        popupElement.style.borderTopColor = "#CCCCCC";
        popupElement.style.borderTopWidth = "1";
        popupElement.style.borderRightStyle = "solid";
        popupElement.style.borderRightColor = "#CCCCCC";
        popupElement.style.borderRightWidth = "1";
        popupElement.style.borderBottomStyle = "solid";
        popupElement.style.borderBottomColor = "#CCCCCC";
        popupElement.style.borderBottomWidth = "1";
        popupElement.style.paddingLeft = "2";
        popupElement.style.paddingTop = "1";
        popupElement.style.paddingRight = "2";
        popupElement.style.paddingBottom = "1";
        popupElement.style.fontSize = "80%";
      }
```

```
    popupElement.style.position = "absolute";
    popupElement.style.visibility = "hidden";
    var tipText = document.createTextNode("");
    popupElement.appendChild(tipText);
    document.body.appendChild(popupElement);
    // Объявляем общедоступные свойства и методы
    this.popupTipStyle = popupElement.style;
    this.enabled = true;
    this.setTip = mjspsPTSetTip;
    this.removeTip = mjspsPTRemoveTip;
    // Список всех всплывающих подсказок
    this.tips = new Array();
    // Объявляем различные служебные свойства
    this.popup = popupElement;
    var piObject = jspsGetProgramInfo();
    this.isIEorOpera = ((piObject.programName == JSPS_GPI_MSIE) ||
    ↳(piObject.programName == JSPS_GPI_OPERA));
    // Номер данного экземпляра объекта в списке jsps_PT_list
    this.myNumber = jsps_PT_counter;
    // Заносим данный экземпляр объекта в список jsps_PT_list
    jsps_PT_list[jsps_PT_counter] = this;
    // Увеличиваем счетчик экземпляров этого объекта на единицу,
    // подготавливая его для создания нового экземпляра
    jsps_PT_counter++;
  }

// Служебная функция, возвращающая индекс элемента массива tips,
// соответствующего переданному ей элементу страницы (параметр pElement)
function jspsPTGetElementIndex(pPTObject, pElement)
  {
    var elementIndex = -1;
    for (var i = 0; i < pPTObject.tips.length; i++)
      if (pPTObject.tips[i][0] == pElement)
        elementIndex = i;
    return elementIndex;
  }

// Функция, реализующая метод setTip
function mjspsPTSetTip(pElement, pTipText)
  {
    // Проверяем, передан ли параметр pTipText, и, если не передан,
    // извлекаем текст подсказки из свойства title
    var isTipInTitle = false;
```

```
  if (!pTipText)
    {
      pTipText = pElement.title;
      // "Позаимствовав" текст подсказки из свойства title, не
      // забудем присвоить этому свойству пустую строку, чтобы подавить
      // вывод всплывающих подсказок самим Web-обозревателем
      pElement.title = "";
      // Признак, был ли текст подсказки взят из свойства title
      isTipInTitle = true;
    }
  // Проверяем, не находится ли уже в массиве tips текст подсказки для
  // данного элемента страницы
  var elementIndex = jspsPTGetElementIndex(this, pElement);
  if (elementIndex == -1)
    {
      // Если подсказка для данного элемента страницы еще не создана,
      // добавляем ее в массив tips
      if (this.isIEorOpera)
        {
          // В случае Internet Explorer и Opera мы сначала сохраняем
          // уже привязанные к событиям onMouseOver и onMouseOut этого
          // элемента страницы обработчики
          var onMouseOverHandler = null;
          var onMouseOutHandler = null;
          if (pElement.onmouseover)
            onMouseOverHandler = pElement.onmouseover;
          if (pElement.onmouseout)
            onMouseOutHandler = pElement.onmouseout;
          // Создаем новый элемент массива tips, который сам является
          // массивом. Вот элементы вложенного массива: элемент
          // страницы, текст подсказки, признак, был ли этот текст
          // взят из свойства title, и сохраненные ранее обработчики
          // событий onMouseOver и onMouseOut
          this.tips[this.tips.length] = new Array(pElement, pTipText,
          ↳isTipInTitle, onMouseOverHandler, onMouseOutHandler);
          // Привязываем обработчики к событиям элемента
          pElement.onmouseover = jspsPTShowPopupTipIEandOpera;
          pElement.onmouseout = jspsPTHidePopupTipIEandOpera;
        }
      else
        {
          // В случае Firefox все несколько проще. Вложенный массив
          // будет содержать только элемент страницы, текст подсказки и
          // признак, был ли этот текст взят из свойства title
```

```
            this.tips[this.tips.length] = new Array(pElement, pTipText,
            ↳isTipInTitle);
            // Привязываем функции-слушатели к событиям элемента
            pElement.addEventListener("mouseover", jspsPTShowPopupTipFF,
            ↳false);
            pElement.addEventListener("mouseout", jspsPTHidePopupTipFF,
            ↳false);
          }
        // Создаем в экземпляре объекта, соответствующем элементу
        // страницы, свойство ptObjectNumber и присваиваем ему
        // порядковый номер экземпляра объекта PopupTip в массиве
        // jsps_PT_list. Он понадобится нам, чтобы получить доступ
        // к экземпляру этого объекта из функций-обработчиков событий
        // и функций-слушателей
        pElement.ptObjectNumber = this.myNumber;
      }
    else
      {
        // Если же к данному элементу страницы уже была привязана
        // подсказка, мы только меняем ее текст
        this.tips[elementIndex][1] = pTipText;
        this.tips[elementIndex][2] = isTipInTitle;
      }
  }

// Функция, реализующая метод removeTip
function mjspsPTRemoveTip(pElement)
  {
    // Получаем индекс элемента массива, где хранится текст подсказки
    // для данного элемента страницы
    var elementIndex = jspsPTGetElementIndex(this, pElement);
    if (elementIndex > -1)
      {
        if (this.isIEorOpera)
          {
            // В случае Internet Explorer и Opera возвращаем на место
            // сохраненные обработчики событий или, если таковых нет,
            // просто их удаляем
            pElement.onmouseover = (this.tips[elementIndex][3]) ?
            ↳this.tips[elementIndex][3] : null;
            pElement.onmouseout = (this.tips[elementIndex][4]) ?
            ↳this.tips[elementIndex][4] : null;
          }
```

```
        else
          {
            // В случае Firefox удаляем функции-слушатели событий
            pElement.removeEventListener("mouseover",
            ↳jspsPTShowPopupTipFF, false);
            pElement.removeEventListener("mouseout",
            ↳jspsPTHidePopupTipFF, false);
          }
        // Если текст подсказки был взят из свойства title, возвращаем
        // его снова в это свойство
        if (this.tips[elementIndex][5])
          pElement.title = this.tips[elementIndex][5];
        // Удаляем элемент массива tips, соответствующий данному элементу
        // страницы
        delete this.tips[elementIndex];
        this.tips.splice(elementIndex, 1);
        // Удаляем созданное ранее свойство ptObjectNumber
        pElement.removeAttribute("ptObjectNumber");
      }
  }

// Функция, выводящая подсказку для Internet Explorer и Opera
function jspsPTShowPopupTipIEandOpera()
  {
    // Получаем экземпляр объекта PopupTip по его порядковому номеру
    var src = event.srcElement;
    var my = jsps_PT_list[src.ptObjectNumber];
    // Получаем индекс элемента массива, где хранится текст подсказки
    // для данного элемента страницы
    var elementIndex = jspsPTGetElementIndex(my, src);
    if (elementIndex > -1)
      {
        var el = my.tips[elementIndex];
        // Вызываем сохраненный обработчик события onMouseOver
        if (el[3]) el[3]();
        if (my.enabled)
          {
            // Выводим подсказку на экран
            my.popup.firstChild.nodeValue = el[1];
            my.popupTipStyle.left = event.clientX + 20;
            // Располагаем всплывающую подсказку под элементом страницы.
            // Свойства offsetTop и offsetHeight объекта,
            // соответствующего элементу страницы, возвращают
```

```
            // вертикальную координату верхнего левого угла
            // этого элемента страницы и его высоту соответственно
            my.popupTipStyle.top = src.offsetTop + src.offsetHeight;
            my.popupTipStyle.visibility = "visible";
          }
      }
  }

// Функция, прячущая подсказку для Internet Explorer и Opera
function jspsPTHidePopupTipIEandOpera()
  {
    // Получаем экземпляр объекта PopupTip по его порядковому номеру
    var src = event.srcElement;
    var my = jsps_PT_list[src.ptObjectNumber];
    // Получаем индекс элемента массива, где хранится текст подсказки
    // для данного элемента страницы
    var elementIndex = jspsPTGetElementIndex(my, src);
    if (elementIndex > -1)
      {
        // Убираем подсказку с экрана
        my.popupTipStyle.visibility = "hidden";
        var el = my.tips[elementIndex];
        // Вызываем сохраненный обработчик события onMouseOut
        if (el[4]) el[4]();
      }
  }

// Функция, выводящая подсказку для Firefox
function jspsPTShowPopupTipFF(event)
  {
    // Получаем экземпляр объекта PopupTip по его порядковому номеру
    var src = event.target;
    var my = jsps_PT_list[src.ptObjectNumber];
    if (my.enabled)
      {
        // Получаем индекс элемента массива, где хранится текст подсказки
        // для данного элемента страницы
        var elementIndex = jspsPTGetElementIndex(my, src);
        if (elementIndex > -1)
          {
            // Выводим подсказку на экран
            my.popup.firstChild.nodeValue = my.tips[elementIndex][1];
            my.popupTipStyle.left = event.clientX + 20;
```

```
            // Располагаем всплывающую подсказку под элементом страницы.
            // Свойства offsetTop и offsetHeight объекта,
            // соответствующего элементу страницы, возвращают
            // вертикальную координату верхнего левого угла
            // этого элемента страницы и его высоту соответственно
            my.popupTipStyle.top = src.offsetTop + src.offsetHeight;
            my.popupTipStyle.visibility = "visible";
          }
      }
  }

// Функция, прячущая подсказку для Firefox
function jspsPTHidePopupTipFF(event)
  {
    // Получаем экземпляр объекта PopupTip по его порядковому номеру
    var src = event.target;
    var my = jsps_PT_list[src.ptObjectNumber];
    // Получаем индекс элемента массива, где хранится текст подсказки
    // для данного элемента страницы
    var elementIndex = jspsPTGetElementIndex(my, src);
    if (elementIndex > -1)
      // Убираем подсказку с экрана
      my.popupTipStyle.visibility = "hidden";
  }
```

Внимание!

Объявление объекта `PopupTip` использует функцию `jspsGetProgramInfo`, чье объявление приведено в листинге 3.1.

Хорошая идея!

Поместите объявление этого объекта в файл сценариев popuptip.js. Впоследствии, чтобы использовать его, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="popuptip.js"></SCRIPT>
```

Пример

Далее приведен HTML-код небольшой Web-страницы с тремя гиперссылками, при наведении курсора мыши на которые выводятся всплывающие подсказки.

```
<HTML>
  <HEAD>
    <TITLE>Всплывающие подсказки</TITLE>
    <SCRIPT SRC="browserdetect.js"></SCRIPT>
```

```
    <!-- Предполагается, что объявление функции jspsGetProgramInfo было
    помещено в файл сценариев browserdetect.js -->
    <SCRIPT SRC="popuptip.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта PopupTip было
    помещено в файл сценариев popuptip.js -->
  </HEAD>
  <BODY>
    <P><A ID="hrf1" HREF="page1.html">Страница 1</A></P>
    <P><A ID="hrf2" HREF="page2.html">Страница 2</A></P>
    <P><A ID="hrf3" HREF="page3.html" TITLE="На последнюю страницу">
    ↳Страница 3</A></P>
    <SCRIPT TYPE="text/javascript">
      var hrf1Object = document.all["hrf1"];
      var hrf2Object = document.all["hrf2"];
      var hrf3Object = document.all["hrf3"];
      var ptObject = new PopupTip();
      ptObject.setTip(hrf1Object, "На страницу 1");
      ptObject.setTip(hrf2Object, "На страницу 2");
      ptObject.setTip(hrf3Object);
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь для первых двух гиперссылок текст подсказок мы задаем явно, методом `setTip` объекта `PopupTip`. Текст подсказки для третьей гиперссылки будет автоматически взят из свойства `title` экземпляра объекта гиперссылки, благо мы задали его в атрибуте `TITLE` тега `<A>` и при вызове метода `setTip` не указали его второй параметр.

Как создать полосу навигации?

Проблема

"Горячие" изображения, гиперссылки со всплывающими подсказками — это, конечно, замечательно. Но старая добрая полоса навигации выглядит куда лучше...

Решение

Что ж, полоса навигации — тоже не бог весть какая сложность. По сути, полоса навигации — это набор "горячих" изображений, помещенных в таблицу. Последовательность действий по ее созданию такова:

1. Готовим набор изображений, необходимых для создания каждого "горячего" изображения — элемента полосы навигации. Для каждого такого эле-

мента изображений должно быть четыре: отображающее обычное состояние, выводящееся при наведении курсора мыши, "нажатое" состояние и "нажатое" состояние при наведении курсора мыши.

2. Создаем таблицу либо из одной строки (для горизонтальной полосы навигации), либо из одного столбца (для вертикальной) и нужного количества ячеек.
3. Создаем в каждой ячейке этой таблицы "горячее" изображение с использованием изображений, подготовленных на первом шаге.
4. Пишем все необходимые Web-сценарии (меняющие изображения и выполняющие переход при щелчках).

> ***Народ предупреждает!***
>
> Всегда следует проверить, все ли изображения подготовлены. Иначе возможно появление в полосе навигации "пустых мест", что выглядит очень неаккуратно.

Для облегчения задачи создания полосы навигации автором был написан универсальный объект `NavBar` (листинг 7.2). Формат вызова его функции-конструктора таков:

```
NavBar(<Массив, описывающий элементы полосы навигации>,
↳<Номер элемента полосы навигации, "нажатого" изначально>
↳[, <Горизонтальная или вертикальная>
↳[, <Стилевой класс, который будет привязан к полосе навигации>]]);
```

Второй параметр задает номер элемента полосы навигации, который должен быть "нажатым" изначально. Обычно изначально "нажатым" делают элемент, соответствующий текущей странице. Не забываем при этом, что элементы полосы навигации нумеруются, начиная с нуля.

Третий, необязательный, параметр указывает, какую полосу навигации мы хотим создать: горизонтальную (значение `false`) или вертикальную (значение `true`). Если он пропущен, полоса навигации будет горизонтальной.

С четвертым, также необязательным, параметром все ясно — он передает в конструктор имя стилевого класса, который будет привязан к полосе навигации (точнее, к создающей ее таблице). Это имя передается в строковом виде. Если оно не указано, никакого стилевого класса к полосе навигации привязано не будет.

А теперь рассмотрим первый параметр. Он передает конструктору массив, описывающий элементы полосы навигации. Каждый элемент этого массива представляет собой другой массив, элементы которого задают следующие параметры соответствующего ему элемента полосы навигации:

- интернет-адрес файла изображения, отображающего обычное состояние данного элемента полосы навигации;

- интернет-адрес файла изображения, выводящегося при наведении на элемент курсора мыши;
- интернет-адрес файла изображения, отображающего "нажатое" состояние данного элемента;
- интернет-адрес файла изображения, выводящегося при наведении на "нажатый" элемент курсора мыши;
- интернет-адрес, по которому будет выполнен переход при щелчке на данном элементе;
- цель предыдущего интернет-адреса. Возможные значения этого параметра приведены в начале текущей главы.

Нужно иметь в виду, что этот массив должен существовать все время, пока существует страница (т. е. быть глобальным). Дело в том, что объект `NavBar` использует его для вывода полосы навигации и, если этот массив оказался недоступным (например, если мы его удалим), не сможет правильно ее отобразить.

Метод `putHere` объекта `NavBar` выводит готовую полосу навигации в то место, где встретилось выражение с его вызовом. Этот метод не принимает параметров и не возвращает значения.

Метод `setPressed` делает элемент полосы навигации, номер которого передан в качестве единственного параметра, "нажатым". Этот метод не возвращает значения.

Листинг 7.2. Объект `NavBar`, создающий на Web-странице полосу навигации

```
// Счетчик созданных к данному времени экземпляров объекта NavBar
var jsps_NB_counter = 0;
// Список всех созданных к данному времени экземпляров объекта NavBar
var jsps_NB_list = new Array();

// Функция-конструктор
function NavBar(elArray, pPressed, pIsVert, pClassName)
  {
    if (!pIsVert) pIsVert = false;
    this.pressed = pPressed;
    this.elements = elArray;
    this.isVert = pIsVert;
    this.className = pClassName;
    this.putHere = mjspsNBPutHere;
    this.setPressed = mjspsNBSetPressed;
```

```
    // Создаем имя для таблицы, которая станет полосой навигации.
    // Это имя пригодится нам потом
    this.id = "NB" + this;
    // Номер данного экземпляра объекта в списке jsps_NB_list
    this.myNumber = jsps_NB_counter;
    // Заносим данный экземпляр объекта в список jsps_NB_list
    jsps_NB_list[jsps_NB_counter] = this;
    // Увеличиваем счетчик экземпляров этого объекта на единицу,
    // подготавливая его для создания нового экземпляра
    jsps_NB_counter++;
  }

// Функция, реализующая метод putHere
function mjspsNBPutHere()
  {
    var s = "<TABLE ID=\"" + this.id.toString() + "\"";
    if (this.className) s += " CLASS=\"" + this.className + "\"";
    s += ">";
    if (!this.isVert) s += "<TR>";
    for (var i = 0; i < this.elements.length; i++)
      {
        var el = this.elements[i];
        if (this.isVert) s += "<TR>";
        s += "<TD><A HREF=\"" + el[4] + "\"";
        if (el[5]) s += " TARGET=\"" + el[5] + "\"";
        s += " ONCLICK=\"jspsNBPressElement(" + this.myNumber.toString()
        ↳+ ", " + i.toString() + ");\">";
        s += "<IMG SRC=\"";
        s += (this.pressed == i) ? el[2] : el[0];
        s += "\"";
        s += " ONMOUSEOVER=\"jspsNBSwapElement(" +
        ↳this.myNumber.toString() + ", " + i.toString() + ", true);\"";
        s += " ONMOUSEOUT=\"jspsNBSwapElement(" +
        ↳this.myNumber.toString() + ", " + i.toString() + ", false);\"";
        s += " BORDER=\"0\"";
        s += "</A></TD>";
        if (this.isVert) s += "</TR>";
      }
    if (!this.isVert) s += "</TR>";
    s += "</TABLE>";
    document.write(s);
  }
```

```
// Служебная функция, возвращающая изображение —
// элемент полосы навигации по его номеру
function jspsNBGetElement(my, elementIndex)
  {
    var nbObject = document.all[my.id];
    if (my.isVert)
      var elObject = nbObject.firstChild.childNodes[elementIndex].
      ↳firstChild.firstChild.firstChild
    else
      var elObject = nbObject.firstChild.firstChild.
      ↳childNodes[elementIndex].firstChild.firstChild;
    return elObject;
  }

// Функция, реализующая метод setPressed
function mjspsNBSetPressed(elementIndex)
  {
    this.pressed = elementIndex;
    for (var i = 0; i < this.elements.length; i++)
      {
        var elObject = jspsNBGetElement(this, i);
        elObject.src = (this.pressed == i) ? this.elements[i][2] :
        ↳this.elements[i][0];
      }
  }

// Функция — обработчик события onClick изображения — элемента полосы
// навигации. Занимается тем, что переключает элемент
// в "нажатое" состояние
function jspsNBPressElement(myNumber, elementIndex)
  {
    var my = jsps_NB_list[myNumber];
    var elObject = jspsNBGetElement(my, elementIndex);
    my.setPressed(elementIndex);
  }

// Функция — обработчик событий onMouseOver и onMouseOut изображения —
// элемента полосы навигации. Фактически реализует "горячее" изображение
function jspsNBSwapElement(myNumber, elementIndex, state)
  {
    var my = jsps_NB_list[myNumber];
    var elObject = jspsNBGetElement(my, elementIndex);
```

```
    if (state)
      elObject.src = (my.pressed == elementIndex) ?
      ↳my.elements[elementIndex][3] : my.elements[elementIndex][1]
    else
      elObject.src = (my.pressed == elementIndex) ?
      ↳my.elements[elementIndex][2] : my.elements[elementIndex][0];
  }
```

Хорошая идея!

Поместите объявление этого объекта в файл сценариев navbar.js. Впоследствии, чтобы использовать его, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="navbar.js"></SCRIPT>
```

Пример

Далее приведены HTML-коды двух Web-страниц, содержащих полосы навигации и с их помощью связанные друг с другом.

Вот HTML-код страницы page1.html:

```
<HTML>
  <HEAD>
    <TITLE>Страница 1</TITLE>
    <SCRIPT SRC="navbar.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта NavBar было
    помещено в файл сценариев navbar.js -->
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var els = new Array();
      els[0] = new Array("1-1.gif", "1-2.gif", "1-3.gif", "1-4.gif",
      "page1.html");
      els[1] = new Array("2-1.gif", "2-2.gif", "2-3.gif", "2-4.gif",
      "page2.html");
      var nbObject = new NavBar(els, 0);
      nbObject.putHere();
    </SCRIPT>
  </BODY>
</HTML>
```

А вот HTML-код страницы page2.html:

```
<HTML>
  <HEAD>
```

```
    <TITLE>Страница 2</TITLE>
    <SCRIPT SRC="navbar.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта NavBar было
    помещено в файл сценариев navbar.js -->
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var els = new Array();
      els[0] = new Array("1-1.gif", "1-2.gif", "1-3.gif", "1-4.gif",
      "page1.html");
      els[1] = new Array("2-1.gif", "2-2.gif", "2-3.gif", "2-4.gif",
      "page2.html");
      var nbObject = new NavBar(els, 1);
      nbObject.putHere();
    </SCRIPT>
  </BODY>
</HTML>
```

Как видим, обе страницы практически идентичны. Единственное существенное отличие — значение второго параметра конструктора объекта `NavBar`. В странице page1.html мы передаем им значение 0, поскольку первый элемент полосы навигации, "отвечающий" за эту страницу, мы хотим сделать изначально "нажатым". (Поскольку в Web-обозреватель загружена страница page1.html, то именно первый элемент полосы навигации и должен быть "нажат", не так ли?) В странице page2.html уже передается значение 1 — мы делаем "нажатым" изначально второй элемент полосы навигации, поскольку именно он ссылается на страницу page2.html. В результате полоса навигации на обеих страницах будет работать корректно.

Попробуем открыть страницу page1.html в Web-обозревателе. Полоса навигации будет благополучно создана, и первый ее элемент, ссылающийся на страницу page1.html, станет "нажатым" по умолчанию. Теперь щелкнем по второму элементу полосы и посмотрим, что получится.

Народ замечает

Пожалуй, явное указание элемента полосы навигации, который должен быть "нажат" изначально, — лучший подход в данном случае.

Можно, конечно, попытаться написать сценарий, который будет извлекать из свойства `pathname` объекта `location`, вложенного в объект `window`, путь и имя текущей страницы и сравнивать его с интернет-адресами, хранящимися в массиве элементов полосы навигации, но это очень сложно и чревато многими проблемами. Как правило, подобные сценарии работают правильно, если страницы загружаются с Web-сервера, но если они открываются с локального компьютера, начинают давать сбои.

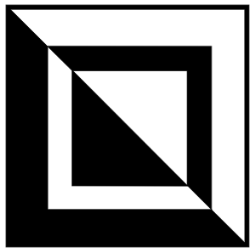
Народ советует

Многие программы Web-редакторов и пакетов интернет-графики позволяют создавать полосы навигации. В частности, популярнейший Web-редактор Macromedia Dreamweaver поможет создать вполне приличную полосу навигации из уже подготовленного набора изображений. А создание полосы навигации в мощном пакете растровой интернет-графики Macromedia Fireworks — совсем простое дело!

Что дальше?

Битва со средствами навигации закончена. И легкие гиперссылки, и тяжеловооруженные полосы навигации повержены. Наши победили.

С поля брани мы попадем прямо за парту. У доски стоит народ и нервно щелкает указкой. Тема урока — таблицы и работа с ними.



ГЛАВА 8

Вывод информации в таблицах

Таблицы, таблицы, таблицы... Огромные, вываливающие на головы читателей уйму текста и цифр. Просто смотреть страшно! Ну и как с ними справиться?! Только и остается, что уповать на помощь неунывающего и поэтому непобедимого народа. Что-то он нам расскажет?..

Как получить доступ к нужному элементу таблицы?

Проблема

Мне нужно из сценария получить доступ к элементу таблицы (строке или ячейке). Как это сделать?

Решение

Универсальное — воспользоваться средствами DOM. В этом случае иерархия вложенных друг в друга объектов будет такова:

- экземпляр объекта — таблица (тег <TABLE>). Самый "верхний" объект во всей табличной иерархии;
- экземпляры объектов, соответствующие заголовку таблицы (тег <CAPTION>), секции "шапки" (тег <THEAD>), секции тела (тег <TBODY>) и секции "поддона" таблицы (тег <TFOOT>). Итого их четыре;
- экземпляры объектов, соответствующие строкам таблицы (тег <TR>). Строки таблицы могут присутствовать только в секциях "шапки", тела и "поддона" таблицы;
- экземпляры объектов, соответствующие ячейкам таблицы (тег <TD> для обычной ячейки и <TH> для ячейки заголовка; такие ячейки часто приме-

няются в "шапке" таблицы). Ячейки таблицы могут присутствовать только в строках;

- экземпляры объектов, соответствующие тегам и фрагментам текста, находящимся в ячейках (в общем, содержимому ячеек). Содержимое ячеек может присутствовать только в тегах <TD>.

> ***Народ предупреждает!***
>
> Не забываем, что на "пути" от таблицы к строкам находятся еще и секции таблицы! Они присутствуют даже в том случае, если мы не зададим их в HTML-коде таблицы явно, с помощью соответствующих тегов; в этом случае в таблице будет неявно присутствовать только секция тела (тег <TBODY>). Технология DOM в смысле правильности HTML-кода очень щепетильна.

Пример 1

Вот HTML-код, создающий на Web-странице небольшую таблицу с тремя столбцами и пятью строками (из них — одна строка "шапки" и одна строка "поддона"), а также заголовком:

```
<TABLE ID="tab">
  <CAPTION>Это таблица</CAPTION>
  <THEAD>
    <TR>
      <TD>Столбец 1</TD>
      <TD>Столбец 2</TD>
      <TD>Столбец 3</TD>
    </TR>
  </THEAD>
  <TBODY>
    <TR>
      <TD>1</TD>
      <TD>2</TD>
      <TD>3</TD>
    </TR>
    <TR>
      <TD>4</TD>
      <TD>5</TD>
      <TD>6</TD>
    </TR>
    <TR>
      <TD>7</TD>
      <TD>8</TD>
      <TD>9</TD>
    </TR>
  </TBODY>
```

```
  <TFOOT>
    <TR>
      <TD>1</TD>
      <TD>2</TD>
      <TD>3</TD>
    </TR>
  </TFOOT>
</TABLE>
```

Создав эту таблицу, мы можем получить доступ к любому ее элементу с помощью свойств DOM.

```
var tabObject = document.all["tab"];
var tabCaption = tabObject.firstChild.firstChild.nodeValue;
```

Этот сценарий помещает в переменную `tabCaption` текст заголовка таблицы. Здесь мы сначала получаем доступ к экземпляру объекта — заголовку таблицы (тегу `<CAPTION>`), потом к экземпляру объекта — фрагменту текста, который помещен в тег `<CAPTION>`, и уже затем обращаемся к свойству `nodeValue` фрагмента текста, чтобы получить сам текст заголовка таблицы.

```
var cell5Text = tabObject.childNodes[2].childNodes[1].childNodes[1].
↳firstChild.nodeValue;
```

А этот сценарий помещает в переменную `cell5Text` текстовое содержимое второй ячейки второй строки секции тела таблицы. Давайте разберем путь к этому текстовому содержимому по частям:

- `tabObject` — экземпляр объекта — таблица;
- `childNodes[2]` — секция тела таблицы (третий по счету потомок экземпляра объекта — таблицы, после заголовка и секции "шапки");
- `childNodes[1]` — вторая строка секции тела таблицы;
- `childNodes[1]` — вторая ячейка второй строки секции тела таблицы;
- `firstChild` — фрагмент текста — содержимое второй ячейки второй строки секции тела таблицы;
- `nodeValue` — значение этого фрагмента текста, т. е. сам текст.

```
tabObject.childNodes[3].firstChild.childNodes[2].firstChild.nodeValue =
↳"Итог по столбцу 3";
```

А этот сценарий поместит в третью ячейку единственной строки секции "поддона" таблицы текст "Итог по столбцу 3".

Пример 2

"Урезанная" модификация предыдущей таблицы:

```
<TABLE ID="tab">
  <TR>
    <TD>1</TD>
    <TD>2</TD>
    <TD>3</TD>
  </TR>
  <TR>
    <TD>4</TD>
    <TD>5</TD>
    <TD>6</TD>
  </TR>
  <TR>
    <TD>7</TD>
    <TD>8</TD>
    <TD>9</TD>
  </TR>
</TABLE>
```

Здесь мы не задали ни заголовок, ни секции таблицы. (Пожалуй, такие таблицы встречаются в практике Web-дизайна чаще всего.)

Вспоминаем, что предупреждал нас народ. Если в таблице не были явно, с помощью соответствующих тегов, заданы секции, секция тела все равно неявно присутствует. Она является единственным потомком экземпляра объекта — таблицы, т. е. тега `<TABLE>`. Значит, чтобы поместить в переменную `cell5Text` текстовое содержимое второй ячейки второй строки секции тела таблицы, мы должны написать такой сценарий:

```
var tabObject = document.all["tab"];
var cell5Text = tabObject.firstChild.childNodes[1].childNodes[1].
↳firstChild.nodeValue;
```

Разберем наш путь по частям:

- `tabObject` — экземпляр объекта — таблица;
- `firstChild` — секция тела таблицы (единственная, неявно присутствующая, если секции не были заданы явно, тегами);
- `childNodes[1]` — вторая строка секции тела таблицы;
- `childNodes[1]` — вторая ячейка второй строки секции тела таблицы;
- `firstChild` — фрагмент текста — содержимое второй ячейки второй строки секции тела таблицы;
- `nodeValue` — значение этого фрагмента текста, т. е. сам текст.

Как создать таблицу программно?

Проблема

Я хочу создать на странице таблицу, причем программно, из Web-сценария. Это сложно?

Решение 1

Самое простое и действенное — создать строку, содержащую HTML-код таблицы, и вывести его на нужное место страницы методом `write` объекта `document`.

Пример

Вот HTML-код страницы, программно выводящей на экран таблицу:

```
<HTML>
  <HEAD>
    <TITLE>Таблица, выводимая программно</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var s = "<TABLE>";
      s += "<CAPTION>Таблица, выводимая программно</CAPTION>";
      s += "<THEAD><TR><TH>Столбец 1</TH><TH>Столбец 2</TH>
      ↳<TH>Столбец 3</TH></TR></THEAD>";
      s += "<TBODY>";
      var t = 1;
      for (var i = 0; i < 3; i++)
        {
          s += "<TR>";
          for (var j = 0; j < 3; j++)
            {
              s += "<TD>" + t.toString() + "</TD>";
              t++;
            }
          s += "</TR>";
        }
      s += "</TBODY>";
      s += "<TFOOT><TR><TD>1</TD><TD>2</TD><TD>3</TD></TR></TFOOT>";
      s += "</TABLE>";
      document.write(s);
    </SCRIPT>
  </BODY>
</HTML>
```

Народ советует

Если нужно перерисовывать созданную таким способом таблицу, нам придется добавить в тег <TABLE> атрибут ID и с его помощью задать имя таблицы. Тогда мы сможем получить доступ к самой этой таблице и изменить ее содержимое с помощью методов и свойств DOM (как это сделать, будет описано далее).

Решение 2

Несравнимо более трудоемкое, использующее методы и свойства DOM. Давайте разберем по шагам наши действия в этом случае.

1. Создаем саму таблицу с помощью метода createElement объекта document. Единственным параметром этому методу нужно передать строку "TABLE" — имя тега, которым создается таблица.
2. Создаем заголовок таблицы, если он нужен. Для этого передадим методу createElement объекта document строку "CAPTION".
3. Создаем текстовый фрагмент, который станет заголовком таблицы. Для этого используем метод createTextNode объекта document и передадим ему в качестве параметра нужный текст.
4. Присоединяем созданный текстовый фрагмент к заголовку таблицы. Для этого используем метод appendChild экземпляра объекта — заголовка таблицы и в качестве параметра передадим ему только что созданный текстовый фрагмент.
5. Присоединяем созданный заголовок таблицы к самой таблице. Для этого используем метод appendChild экземпляра объекта — таблицы и в качестве параметра передадим ему только что созданный экземпляр объекта — заголовок.
6. Создаем секцию таблицы. Для этого методу createElement объекта document нужно передать строку, соответствующую имени тега, с помощью которого создается эта секция ("THEAD", "TBODY" или "TFOOT").
7. Присоединяем секцию к таблице, используя метод appendChild экземпляра объекта — таблицы. В качестве параметра передадим этому методу только что созданный экземпляр объекта — секцию таблицы.
8. Создаем строку этой секции таблицы. Параметр метода createElement объекта document — строка "TR".
9. Присоединяем строку к секции таблицы, используя метод appendChild экземпляра объекта — секции. В качестве параметра передадим этому методу только что созданный экземпляр объекта — строку.
10. Создаем ячейку этой секции таблицы. Параметр метода createElement объекта document — строка "TD" для обычной ячейки или "TH" для ячейки заголовка.

11. Присоединяем ячейку к строке методом `appendChild` экземпляра объекта — строки. В качестве параметра передадим этому методу только что созданный экземпляр объекта — ячейку.
12. Создаем содержимое ячейки. Это может быть иерархия тегов или простой текстовый фрагмент.
13. Присоединяем содержимое ячейки к самой ячейке в качестве ее содержимого. Для этого вызываем метод `appendChild` экземпляра объекта — ячейки с параметром, соответствующим созданному содержимому этой ячейки (фрагменту текста или тегу, создающему это содержимое).
14. Если это не последняя ячейка строки, переходим к шагу 10.
15. Если это не последняя строка секции таблицы, переходим к шагу 8.
16. Если это не последняя секция таблицы, переходим к шагу 6.
17. Присоединяем полностью созданную таблицу к телу страницы. Для этого нужно вызвать метод `appendChild` экземпляра объекта — тела страницы и передать ему экземпляр объекта — созданную нами таблицу.

Народ советует

Если мы хотим создать таблицу исключительно средствами DOM, лучше всего создать явно все ее секции (по крайней мере, секцию тела таблицы). Так мы будем уверены, что все свойства DOM для доступа к элементам таблицы, описанные выше, сработают правильно.

Народ замечает

Да, способ создания таблицы с помощью методов и свойств DOM очень сложен. Но это единственный способ, позволяющий нам впоследствии изменить содержимое таблицы программно без перезагрузки страницы.

Пример

Перепишем приведенный ранее пример страницы, выводящей на экран таблицу, с использованием только средств DOM.

```
<HTML>
  <HEAD>
    <TITLE>Таблица, выводимая программно</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var tableObject = document.createElement("TABLE");
      var captionObject = document.createElement("CAPTION");
      var captionTextObject =
      ↳document.createTextNode("Таблица, выводимая программно");
```

```
captionObject.appendChild(captionTextObject);
tableObject.appendChild(captionObject);
var headSectionObject = document.createElement("THEAD");
var rowObject = document.createElement("TR");
var cellObject = null;
var cellTextObject = null;
for (var i = 1; i < 4; i++)
  {
    cellObject = document.createElement("TH");
    cellTextObject = document.createTextNode("Столбец " +
    ↳i.toString());
    cellObject.appendChild(cellTextObject);
    rowObject.appendChild(cellObject);
  }
headSectionObject.appendChild(rowObject);
tableObject.appendChild(headSectionObject);
var bodySectionObject = document.createElement("TBODY");
var t = 1;
for (i = 0; i < 3; i++)
  {
    rowObject = document.createElement("TR");
    for (var j = 0; j < 3; j++)
      {
        cellObject = document.createElement("TD");
        cellTextObject = document.createTextNode(t.toString());
        cellObject.appendChild(cellTextObject);
        rowObject.appendChild(cellObject);
        t++;
      }
    bodySectionObject.appendChild(rowObject);
  }
tableObject.appendChild(bodySectionObject);
var footSectionObject = document.createElement("TFOOT");
rowObject = document.createElement("TR");
for (i = 1; i < 4; i++)
  {
    cellObject = document.createElement("TD");
    cellTextObject = document.createTextNode(i.toString());
    cellObject.appendChild(cellTextObject);
    rowObject.appendChild(cellObject);
  }
footSectionObject.appendChild(rowObject);
tableObject.appendChild(footSectionObject);
```

```
      document.body.appendChild(tableObject);
    </SCRIPT>
  </BODY>
</HTML>
```

Решение 3

Использование универсального объекта `DynamicTable` (листинг 8.1). Функция-конструктор этого объекта имеет очень простой формат вызова:

```
DynamicTable(<Элемент страницы>[, <Имя таблицы>
↳[, <Стилевой класс для таблицы>]]);
```

Первый, единственный обязательный, параметр задает элемент страницы, в котором будет выведена таблица и чьим потомком она станет.

Вторым, необязательным, параметром в конструктор передается имя создаваемой таблицы. Это имя передается в строковом виде. Если оно не указано, имя таблицы будет сформировано самим объектом `DynamicTable`.

Третьим, также необязательным, параметром передается имя стилевого класса, который будет привязан к создаваемой таблице. Это имя передается в строковом виде. Если оно не указано, никакого стилевого класса к таблице привязано не будет.

Метод `createTable` объекта `DynamicTable` формирует таблицу на основе трех массивов, переданных ему в качестве параметров, сохраняет ее в памяти компьютера, но не выводит на экран. Вот формат вызова этого метода:

```
createTable(<Массив секции тела>[, <Массив секции "шапки">
↳[, <Массив секции "поддона">[, <Текст заголовка таблицы>
[, <Индекс первой строки>, <Индекс последней строки>]]]]);
```

Переданные этому методу три массива задают содержимое, соответственно, секции тела, "шапки" и "поддона" таблицы. Если какой-либо из этих массивов не указан или вместо него было передано значение `null`, данная секция в таблице создана не будет. Также можно указать *Текст заголовка таблицы*.

Все три массива, передаваемые методу `createTable`, представляют строки соответствующей секции создаваемой таблицы. Каждый элемент такого массива, в свою очередь, содержит вложенный массив, который представляет ячейки данной строки таблицы. Элементы этого вложенного массива могут содержать:

- экземпляр объекта, который станет содержимым ячейки;
- строку или число, которые станут содержимым ячейки;
- пустую строку или пробел, если нужно создать пустую ячейку;
- одну из псевдоконстант JavaScript, задающих специальные значения.

Этих псевдоконстант предусмотрено две (их значения можно увидеть в листинге 8.1):

- JSPS_DT_SPANNED_HORZ — данная ячейка будет объединена с ячейкой, расположенной левее;
- JSPS_DT_SPANNED_VERT — данная ячейка будет объединена с ячейкой, расположенной выше.

Количество элементов всех вложенных массивов (тех, что представляют ячейки таблицы) должно быть одинаковым. В противном случае мы получим сообщение об ошибке, и таблица не будет создана.

Мы можем вывести в таблицу не весь *Массив секции тела*, а его часть. Для этого достаточно указать *Индекс первой строки* и *Индекс последней строки*, соответствующие элементам массива, которые должны присутствовать в таблице. Не забываем, что индексы элементов массивов нумеруются, начиная с нуля.

Если мы хотим создать таблицу на основе данных, которые нигде не хранятся, а, скажем, вычисляются в Web-сценарии, то воспользуемся методом createTableAdvanced объекта DynamicTable. Этот метод также формирует таблицу в памяти без вывода ее на экран. Вот формат его вызова:

```
createTableAdvanced(<Функция, формирующая содержимое таблицы>,
<Количество строк секции тела>, <Количество столбцов>
[, <Количество строк секции "шапки">, <Количество строк секции "поддона">
[, <Индекс первой строки>, <Индекс последней строки>
[, <Дополнительный параметр функции, формирующей содержимое таблицы>]]]);
```

Функция, формирующая содержимое таблицы, переданная в этот метод, должна принимать четыре параметра: номер строки таблицы, номер ячейки в этой строке (нумерация и строк, и ячеек начинается с нуля — не забываем об этом!), значение, задающее секцию таблицы, и *Дополнительный параметр*, заданный в вызове метода createTableAdvanced.

Значения, задающие секцию таблицы, в виде псевдоконстант JavaScript перечислены далее (их значения можно увидеть в листинге 8.1):

- JSPS_DT_TS_CAPTION — заголовок таблицы;
- JSPS_DT_TS_HEAD — секция "шапки" таблицы;
- JSPS_DT_TS_BODY — секция тела таблицы;
- JSPS_DT_TS_FOOT — секция "поддона" таблицы.

Функция, формирующая содержимое таблицы, должна возвращать значение одного из четырех типов:

- экземпляр объекта, который станет содержимым ячейки;
- строку или число, которые станут содержимым ячейки;

- пустую строку или пробел, если нужно создать пустую ячейку;
- псевдоконстанту JSPS_DT_SPANNED_HORZ или JSPS_DT_SPANNED_VERT (их назначение было описано ранее).

Также обязательно нужно задать *Общее количество строк* и *Количество столбцов* таблицы. Иначе объект DynamicTable не будет знать, какого размера таблицу мы хотим создать. (В случае метода createTable количество строк известно — оно равно сумме элементов всех массивов. Также известно количество столбцов — оно равно количеству элементов вложенных массивов.) Также мы можем задать *Количество строк секции "шапки"* и *Количество строк секции "поддона"*, если хотим создать эти секции.

И в этом случае мы можем вывести в таблицу (точнее, в ее секцию тела) не все приготовленные данные, а их часть. Для этого следует указать *Индекс первой строки* и *Индекс последней строки*, которые должны присутствовать в таблице.

Метод showTable объекта DynamicTable выводит сформированную в памяти таблицу на экран. Этот метод не принимает параметров и не возвращает результата.

Метод removeTable объекта DynamicTable удаляет таблицу с экрана и из памяти компьютера, после чего она становится недоступной. Этот метод не принимает параметров, не возвращает результата и пригодится, если мы хотим создать заново таблицу на основе других данных.

Свойство id объекта DynamicTable предоставляет доступ к имени таблицы. Оно будет полезно, если мы не указали имя таблицы при вызове конструктора этого объекта, и объект сам сформировал для нее имя.

А свойство table объекта DynamicTable предоставляет доступ к самой таблице, сформированной в памяти и, возможно, уже выведенной на экран методом showTable. Используя это свойство, мы можем добраться до самой таблицы и ее элементов, а также использовать готовую таблицу для других нужд (например, поместить ее в ячейку другой таблицы).

Листинг 8.1. Объект DynamicTable, выводящий на экран таблицу

```
var JSPS_DT_SPANNED_HORZ = "!!!JSPS_DT_SPANNED_HORZ!!!";
var JSPS_DT_SPANNED_VERT = "!!!JSPS_DT_SPANNED_VERT!!!";

var JSPS_DT_TS_CAPTION = 0;
var JSPS_DT_TS_HEAD = 1;
var JSPS_DT_TS_BODY = 2;
var JSPS_DT_TS_FOOT = 3;
```

```
// Функция-конструктор
function DynamicTable(pElement, pID, pClassName)
  {
    this.element = pElement;
    this.id = (pID) ? pID : "tab" + this;
    this.table = null;
    this.className = (pClassName) ? pClassName : "";
    this.createTable = mjspsDTCreateTable;
    this.createTableAdvanced = mjspsDTCreateTableAdvanced;
    this.showTable = mjspsDTShowTable;
    this.removeTable = mjspsDTRemoveTable;
    this.createTableTag = mjspsDTCreateTableTag;
    this.createTableCaption = mjspsDTCreateTableCaption;
    this.createTableSection = mjspsDTCreateTableSection;
  }

// Функция, реализующая служебный метод createTableTag.
// Этот метод создает саму таблицу
function mjspsDTCreateTableTag()
  {
    this.table = document.createElement("TABLE");
    this.table.id = this.id;
    if (this.className)
      this.table.className = this.className;
  }

// Служебная функция, формирующая содержимое таблицы и используемая при
// вызове метода createTable. (При вызове метода createTableAdvanced
// функция, формирующая содержимое таблицы, объявляется самим
// программистом)
function jspsDTGetCellValue(pRowNumber, pColNumber, pSectionCode,
↳pAdditional)
  {
    if (pSectionCode == JSPS_DT_TS_CAPTION)
      return pAdditional
    else
      return pAdditional[pRowNumber][pColNumber];
  }

// Функция, реализующая служебный метод createTableCaption. Этот метод
// создает заголовок таблицы. Он принимает в качестве параметров функцию,
// формирующую содержимое таблицы, и дополнительный параметр
function mjspsDTCreateTableCaption(pFunction, pAdditional)
```

```
  {
    var captionText = pFunction(0, 0, JSPS_DT_TS_CAPTION, pAdditional);
    if (captionText)
      {
        var captionObject = document.createElement("CAPTION");
        var captionTextObject = document.createTextNode(captionText);
        captionObject.appendChild(captionTextObject);
        this.table.appendChild(captionObject);
      }
  }

// Функция, реализующая служебный метод createTableSection. Этот метод
// создает секцию таблицы и заполняет ее данными. Он принимает такие
// параметры: код секции, функцию, формирующую содержимое таблицы,
// количество строк секции, количество столбцов, номер первого выводимого
// элемента, номер последнего выводимого элемента и дополнительный
// параметр
function mjspsDTCreateTableSection(pSectionCode, pFunction, pRowCount,
↳pCellCount, pFirstElementIndex, pLastElementIndex, pAdditional)
  {
    if (pRowCount > 0)
      {
        if (typeof(pFirstElementIndex) == "undefined")
          pFirstElementIndex = 0;
        if (typeof(pLastElementIndex) == "undefined")
          pLastElementIndex = pRowCount - 1;
        switch (pSectionCode)
          {
            case JSPS_DT_TS_HEAD:
              var sectionTag = "THEAD";
              var cellTag = "TH";
              break;
            case JSPS_DT_TS_BODY:
              var sectionTag = "TBODY";
              var cellTag = "TD";
              break;
            case JSPS_DT_TS_FOOT:
              var sectionTag = "TFOOT";
              var cellTag = "TD";
              break;
          }
        var sectionObject = document.createElement(sectionTag);
        var rowObject = null;
```

```
var cellObject = null;
var cellContentObject = null;
var leftCell = null;
var horzSpannedCellCount = 1;
var topCells = new Array();
var vertSpannedCellCounts = new Array();
for (var i = pFirstElementIndex; i <= pLastElementIndex; i++)
  {
    rowObject = document.createElement("TR");
    for (var j = 0; j < pCellCount; j++)
      {
        cellObject = document.createElement(cellTag);
        var cellContent = pFunction(i, j, pSectionCode,
        ↳pAdditional);
        if (cellContent == JSPS_DT_SPANNED_HORZ)
          {
            horzSpannedCellCount++;
            leftCell.colSpan = horzSpannedCellCount;
          }
        else
          if (cellContent == JSPS_DT_SPANNED_VERT)
            {
              vertSpannedCellCounts[j]++;
              topCells[j].rowSpan = vertSpannedCellCounts[j];
            }
          else
            {
              switch (typeof(cellContent))
                {
                  case "object":
                    cellContentObject = cellContent;
                    break;
                  case "string":
                    if (cellContent == "") cellContent = " ";
                    cellContentObject =
                    ↳document.createTextNode(cellContent);
                    break;
                  case "number":
                    cellContentObject =
                    ↳document.createTextNode
                    ↳(cellContent.toString());
                    break;
```

```
                        default:
                          cellContentObject =
                          ↳document.createTextNode(" ");
                          break;
                      }
                    leftCell = cellObject;
                    horzSpannedCellCount = 1;
                    topCells[j] = cellObject;
                    vertSpannedCellCounts[j] = 1;
                    cellObject.appendChild(cellContentObject);
                    rowObject.appendChild(cellObject);
                  }
              }
            sectionObject.appendChild(rowObject);
          }
        this.table.appendChild(sectionObject);
      }
  }

// Функция, реализующая метод createTable
function mjspsDTCreateTable(pBodyArray, pHeadArray, pFootArray, pCaption,
↳pFirstElementIndex, pLastElementIndex)
  {
    this.removeTable();
    this.createTableTag();
    // Заметим, что в этом случае методам createTableCaption и
    // createTableSection мы передаем объявленную ранее служебную функцию
    // jspsDTGetCellValue
    if (pCaption)
      this.createTableCaption(jspsDTGetCellValue, pCaption);
    if (pHeadArray)
      this.createTableSection(JSPS_DT_TS_HEAD, jspsDTGetCellValue,
      ↳pHeadArray.length, pHeadArray[0].length, 0, pHeadArray.length - 1,
      ↳pHeadArray);
    this.createTableSection(JSPS_DT_TS_BODY, jspsDTGetCellValue,
    ↳pBodyArray.length, pBodyArray[0].length, pFirstElementIndex,
    ↳pLastElementIndex, pBodyArray);
    if (pFootArray)
      this.createTableSection(JSPS_DT_TS_FOOT, jspsDTGetCellValue,
      pFootArray.length, pFootArray[0].length, 0, pFootArray.length - 1,
      ↳pFootArray);
  }
```

```
// Функция, реализующая метод createTableAdvanced
function mjspsDTCreateTableAdvanced(pFunction, pBodyRowCount, pCellCount,
↳pHeadRowCount, pFootRowCount, pFirstElementIndex, pLastElementIndex,
↳pAdditional)
  {
    this.removeTable();
    this.createTableTag();
    this.createTableCaption(pFunction);
    if (pHeadRowCount > 0)
      this.createTableSection(JSPS_DT_TS_HEAD, pFunction, pHeadRowCount,
      ↳pCellCount);
    this.createTableSection(JSPS_DT_TS_BODY, pFunction, pBodyRowCount,
    ↳pCellCount, pFirstElementIndex, pLastElementIndex);
    if (pFootRowCount > 0)
      this.createTableSection(JSPS_DT_TS_FOOT, pFunction, pFootRowCount,
      ↳pCellCount);
  }

// Функция, реализующая метод showTable
function mjspsDTShowTable()
  {
    if (this.table)
      this.element.appendChild(this.table);
  }

// Функция, реализующая метод removeTable
function mjspsDTRemoveTable()
  {
    if (this.table)
      {
        this.element.removeChild(this.table);
        this.table = null;
      }
  }
```

Хорошая идея!

Поместите объявление этого объекта в файл сценариев dynamictable.js. Впоследствии, чтобы использовать его, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="dynamictable.js"></SCRIPT>
```

Пример 1

Далее приведен HTML-код Web-страницы, выводящей на экран таблицу — список языков программирования.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="dynamictable.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта DynamicTable помещено в
    файл сценария dynamictable.js -->
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var JSPS_COMPILING = "Компилируемый";
      var JSPS_INTERPRETING = "Интерпретируемый";
      var tableHead = new Array();
      tableHead[0] = new Array("Категория", "Язык");
      var tableBody = new Array();
      tableBody[0] = new Array(JSPS_COMPILING, "C++");
      tableBody[1] = new Array(JSPS_INTERPRETING, "JavaScript");
      tableBody[2] = new Array(JSPS_COMPILING, "Pascal");
      tableBody[3] = new Array(JSPS_COMPILING, "Java");
      tableBody[4] = new Array(JSPS_INTERPRETING, "Perl");
      tableBody[5] = new Array(JSPS_COMPILING, "Delphi");
      tableBody[6] = new Array(JSPS_COMPILING, "Visual Basic");
      var tableObject = new DynamicTable(document.body);
      tableObject.createTable(tableBody, tableHead, null,
      ↳"Языки программирования");
      tableObject.showTable();
    </SCRIPT>
  </BODY>
</HTML>
```

Таким образом, мы создали простейшую таблицу, данные которой извлекаются из двух массивов: `tableHead` (секция "шапки") и `tableBody` (секция тела). Причем создали мы ее минимальными усилиями, воспользовавшись объектом `DynamicTable`.

> ***Народ советует***
>
> Код, создающий массивы, на основе которых будет сформирована таблица, можно вынести в файл сценариев. Таким образом, мы получим возможность править исходные данные таблицы, не затрагивая файлы с HTML-кодом страниц.

Пример 2

Честно говоря, полученная нами в предыдущем примере таблица не очень красива. Так, лучше сделать столбец **Категория** вторым по счету и объединить его ячейки, содержащие одинаковый текст. Давайте перепишем приве-

денный ранее код, чтобы он это делал. Заодно привяжем к создаваемой таблице стиль, чтобы она выглядела приличнее.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="dynamictable.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта DynamicTable помещено в
    файл сценария dynamictable.js -->
    <STYLE>
      .mytable { border: thin solid #CCCCCC; }
      .mytable TD { border: thin solid #CCCCCC; }
    </STYLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var tableBody = new Array();
      tableBody[0] = new Array(0, "C++");
      tableBody[1] = new Array(1, "JavaScript");
      tableBody[2] = new Array(0, "Pascal");
      tableBody[3] = new Array(0, "Java");
      tableBody[4] = new Array(1, "Perl");
      tableBody[5] = new Array(0, "Delphi");
      tableBody[6] = new Array(0, "Visual Basic");
      var tableObject = new DynamicTable(document.body, null, "mytable");
      var secondColValue = -1;
      function constructTable(pRowNumber, pColNumber, pSectionCode)
        {
          switch (pSectionCode)
            {
              case JSPS_DT_TS_CAPTION:
                return "Языки программирования";
                break;
              case JSPS_DT_TS_HEAD:
                return (pColNumber == 1) ? "Категория" : "Язык";
                break;
              case JSPS_DT_TS_BODY:
                if (pColNumber == 1)
                  {
                    var colValue = tableBody[pRowNumber][0];
                    // Проверяем, не хранит ли верхняя ячейка второго
                    // столбца то же самое значение, и, если так,
                    // возвращаем псевдоконстанту JSPS_DT_SPANNED_VERT,
                    // чтобы выполнить объединение ячеек по вертикали
```

```
                    if (secondColValue == colValue)
                      return JSPS_DT_SPANNED_VERT
                    else
                      {
                        secondColValue = colValue;
                        return (colValue == 0) ? "Компилируемый" :
                        ↳"Интерпретируемый";
                      }
                  }
                else
                  return tableBody[pRowNumber][1]
                break;
              case JSPS_DT_TS_FOOT:
                return null;
                break;
            }
        }
      tableObject.createTableAdvanced(constructTable, tableBody.length,
      ↳tableBody[0].length, 1);
      tableObject.showTable();
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь для формирования таблицы мы использовали метод `createTableAdvanced` объекта `DynamicTable`. Это позволило нам, во-первых, изменить порядок столбцов таблицы без правки массива `tableBody`, во-вторых, создать секцию "шапки" таблицы без использования массива, в-третьих, выполнить объединение ячеек второго столбца, хранящих одинаковые значения. Конечно, пришлось повозиться, но результат того стоит.

Как выполнить постраничный вывод таблицы?

Проблема

Мне приходится выводить на Web-страницы очень большие таблицы. Можно ли как-то выводить их не целиком, а по частям, причем, так, чтобы посетитель мог их "листать" без перезагрузки страницы?

Решение

С объявленным в листинге 8.1 объектом `DynamicTable` это делается очень просто. Давайте выясним, как именно.

Сначала нам нужно объявить переменную. В ней будет храниться номер самой верхней строки таблицы, которая в данный момент отображается на экране. Установим изначальное значение этой переменной в ноль, чтобы вывести первую "страницу" таблицы.

Следующий наш шаг — создание таблицы с помощью объекта `DynamicTable`. В качестве параметров метода `createTable` или `createTableAdvanced` этого объекта, устанавливающих номера первой и последней выводимой строк, мы зададим значение объявленной ранее переменной и то же самое значение, но увеличенное на количество одновременно выводимых строк (размер "страницы" таблицы), или оставшееся количество строк таблицы (если оно меньше размера "страницы"). И выведем таблицу на экран.

Чтобы переместиться на следующую "страницу" таблицы, сделаем следующее. Сначала проверим, не последняя ли это "страница" таблицы. Для этого достаточно вычесть из общего размера таблицы номер верхней выводимой строки (значение объявленной ранее переменной) и единицу и сравнить полученную разность с размером "страницы". Если она меньше размера "страницы", значит, это последняя "страница" таблицы, и мы ничего не делаем. В противном случае увеличим значение объявленной ранее переменной на размер "страницы" или на количество оставшихся строк таблицы (если оно меньше размера "страницы") и выведем таблицу на экран.

Для перемещения на предыдущую "страницу" таблицы нужно сделать вот что. Проверяем, не равен ли номер верхней выводимой строки (значение объявленной ранее переменной) нулю, т. е. не первая ли это "страница". Если это так, мы ничего не делаем, иначе уменьшим значение объявленной ранее переменной на размер "страницы" или на количество оставшихся до начала таблицы строк (если оно меньше размера "страницы") и выведем таблицу на экран.

Переместиться на первую "страницу" таблицы проще всего. Присваиваем объявленной ранее переменной ноль и выводим таблицу на экран.

Переместиться на последнюю "страницу" таблицы несколько сложнее. Сначала мы получим номер последней строки таблицы, потом вычтем из него размер "страницы" в строках и занесем полученное значение в переменную, объявленную ранее. После этого останется только вывести таблицу.

> ***Народ советует***
>
> Для "листания" таблицы мы можем использовать гиперссылки, кнопки или что-то более мудреное (например, полосу навигации).

Здесь мы не рассматриваем случай, когда таблица содержит меньше строк, чем должно выводиться в одной ее "странице". Разумеется, тогда вообще не стоило городить огород с постраничным выводом.

Народ советует

Постраничный вывод — неплохое решение для очень больших таблиц, содержащих множество строк. Также можно разбить такие таблицы по какому-то признаку на сегменты разного размера и выводить их посегментно. Но это требует несколько более сложного программирования.

Пример 1

Давайте рассмотрим HTML-код страницы с таблицей, содержимое которой выводится "страницами" фиксированного размера в две строки.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="dynamictable.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта DynamicTable помещено в
    файл сценария dynamictable.js -->
  </HEAD>
  <BODY>
    <!-- Наша таблица будет выведена здесь, в созданном с помощью тега
    <DIV> контейнере -->
    <DIV ID="tableHolder"></DIV>
    <SCRIPT TYPE="text/javascript">
      var JSPS_COMPILING = "Компилируемый";
      var JSPS_INTERPRETING = "Интерпретируемый";
      // Задаем размер "страницы"
      var JSPS_PAGE_SIZE = 2;
      // Задаем номер первой выводимой строки в начальной "странице"
      var topRowNumber = 0;
      var tableHead = new Array();
      tableHead[0] = new Array("Категория", "Язык");
      var tableBody = new Array();
      tableBody[0] = new Array(JSPS_COMPILING, "C++");
      tableBody[1] = new Array(JSPS_INTERPRETING, "JavaScript");
      tableBody[2] = new Array(JSPS_COMPILING, "Pascal");
      tableBody[3] = new Array(JSPS_COMPILING, "Java");
      tableBody[4] = new Array(JSPS_INTERPRETING, "Perl");
      tableBody[5] = new Array(JSPS_COMPILING, "Delphi");
      tableBody[6] = new Array(JSPS_COMPILING, "Visual Basic");
      // При создании экземпляра объекта DynamicTable не забываем
      // указать, что таблица должна выводиться в контейнере
      var tableObject = new DynamicTable(document.all["tableHolder"]);
      // Выводим первую "страницу" таблицы
      putTable(0);
```

```
      // Эта функция выводит таблицу на экран. В качестве параметра она
      // принимает номер первой выводимой строки в текущей "странице"
      function putTable(pFirstRowNumber)
        {
          if (pFirstRowNumber < 0) pFirstRowNumber = 0;
          if (pFirstRowNumber > tableBody.length - 1)
            pFirstRowNumber = tableBody.length - 1;
          topRowNumber = pFirstRowNumber;
          var lastRowNumber = pFirstRowNumber + JSPS_PAGE_SIZE - 1;
          if (lastRowNumber > tableBody.length - 1)
            lastRowNumber = tableBody.length - 1;
          tableObject.createTable(tableBody, tableHead, null,
          ↳"Языки программирования", pFirstRowNumber, lastRowNumber);
          tableObject.showTable();
        }

      // Четыре функции, выполняющие "листание" таблицы
      function toFirst()
        {
          putTable(0);
        }

      function toPrevious()
        {
          putTable(topRowNumber - JSPS_PAGE_SIZE);
        }

      function toNext()
        {
          putTable(topRowNumber + JSPS_PAGE_SIZE);
        }

      function toLast()
        {
          putTable(tableBody.length - JSPS_PAGE_SIZE);
        }
    </SCRIPT>
    <!-- Набор гиперссылок для "листания" таблицы -->
    <P><A HREF="#" ONCLICK="toFirst();">В начало</A> 
    ↳<A HREF="#" ONCLICK="toPrevious();">На предыдущую</A> 
    ↳<A HREF="#" ONCLICK="toNext();">На следующую</A> 
    ↳<A HREF="#" ONCLICK="toLast();">В конец</A></P>
  </BODY>
</HTML>
```

Народ советует

Чтобы вывести таблицу (или любой другой элемент страницы, формируемый программно) строго в определенное место, проще всего вставить в это место контейнер HTML, создаваемый с помощью тега <DIV>, и выводить таблицу в нем. Автор книги так и поступил в приведенном выше примере.

Пример 2

Последуем доброму совету народа и разобьем представленную в предыдущем примере таблицу на сегменты. В качестве критерия разбиения возьмем значение первого столбца таблицы — категорию языков программирования.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="dynamictable.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта DynamicTable помещено в
    файл сценария dynamictable.js -->
  </HEAD>
  <BODY>
    <!-- Наша таблица будет выведена здесь, в созданном с помощью тега
    <DIV> контейнере -->
    <DIV ID="tableHolder"></DIV>
    <SCRIPT TYPE="text/javascript">
      var JSPS_COMPILING = "Компилируемый";
      var JSPS_INTERPRETING = "Интерпретируемый";
      var tableHead = new Array();
      tableHead[0] = new Array("Категория", "Язык");
      var tableBody = new Array();
      tableBody[0] = new Array(JSPS_COMPILING, "C++");
      tableBody[1] = new Array(JSPS_INTERPRETING, "JavaScript");
      tableBody[2] = new Array(JSPS_COMPILING, "Pascal");
      tableBody[3] = new Array(JSPS_COMPILING, "Java");
      tableBody[4] = new Array(JSPS_INTERPRETING, "Perl");
      tableBody[5] = new Array(JSPS_COMPILING, "Delphi");
      tableBody[6] = new Array(JSPS_COMPILING, "Visual Basic");
      // Элементы этого массива будут содержать номера первых строк
      // соответствующих сегментов таблицы
      var segmentTopRowNumbers = new Array();
      // А элементы этого массива будут содержать количества строк
      // соответствующих сегментов таблицы
      var segmentRowCounts = new Array();
      // Выполняем разбивку таблицы на сегменты и заносим в объявленные
      // выше массивы нужные величины
```

```
var category = "";
var segmentNumber = -1;
for (var i = 0; i < tableBody.length; i++)
  {
    var cat = tableBody[i][0];
    if (category == cat)
      segmentRowCounts[segmentNumber]++
    else
      {
        segmentNumber++;
        segmentTopRowNumbers[segmentNumber] = i;
        segmentRowCounts[segmentNumber] = 1;
        category = cat;
      }
  }
// При создании экземпляра объекта DynamicTable не забываем
// указать, что таблица должна выводиться в контейнере
var tableObject = new DynamicTable(document.all["tableHolder"]);
// Задаем номер первого выводимого сегмента
var currentSegmentNumber = 0;
// Выводим первую "страницу" таблицы
putTable(0);

// Эта функция выводит сегмент таблицы на экран. В качестве
// параметра она принимает номер выводимого сегмента
function putTable(pSegmentNumber)
  {
    if (pSegmentNumber < 0) pSegmentNumber = 0;
    if (pSegmentNumber > segmentTopRowNumbers.length - 1)
      pSegmentNumber = segmentTopRowNumbers.length - 1;
    currentSegmentNumber = pSegmentNumber;
    tableObject.createTable(tableBody, tableHead, null,
    ↳"Языки программирования",
    ↳segmentTopRowNumbers[pSegmentNumber],
    ↳segmentTopRowNumbers[pSegmentNumber] +
    ↳segmentRowCounts[pSegmentNumber] - 1);
    tableObject.showTable();
  }

// Четыре функции "листания" таблицы
function toFirst()
  {
    putTable(0);
  }
```

```
      function toPrevious()
        {
          putTable(currentSegmentNumber - 1);
        }

      function toNext()
        {
          putTable(currentSegmentNumber + 1);
        }

      function toLast()
        {
          putTable(segmentTopRowNumbers.length - 1);
        }
    </SCRIPT>
    <!-- Набор гиперссылок для "листания" таблицы -->
    <P><A HREF="#" ONCLICK="toFirst();">В начало</A> 
    ↳<A HREF="#" ONCLICK="toPrevious();">На предыдущую</A> 
    ↳<A HREF="#" ONCLICK="toNext();">На следующую</A> 
    ↳<A HREF="#" ONCLICK="toLast();">В конец</A></P>
  </BODY>
</HTML>
```

Народ советует

Сегментация весьма полезна в справочниках, прайс-листах и т. п., в общем, во всех больших таблицах, содержимое которых разбивается на несколько категорий. Только неплохо было бы сначала сгруппировать строки таблиц по этим самым категориям, чем автор этой книги так и не озаботился.

Как отсортировать строки в таблице?

Проблема

Мне нужно отсортировать строки в таблице по заданному посетителем критерию. Как это сделать?

Решение

Увы — универсальных решений здесь нет. Хотя, если наша таблица построена на основе данных, хранящихся в массиве (например, она была создана методом `createTable` объекта `DynamicTable`, объявление которого представлено в листинге 8.1), можно сначала отсортировать сам массив, а потом уже формировать на его основе таблицу.

Отсортировать массив можно, вызвав метод `sort` объекта `Array`. Формат вызова этого метода таков:

```
<Массив>.sort([<Функция сортировки>]);
```

Если *Функция сортировки* не указана, метод `sort` выполняет простейшую сортировку. Он извлекает два соседних элемента массива, преобразует их в строковый вид и сравнивает. (Опытные компьютерщики говорят, что этот метод выполняет *ASCII-сортировку*.) После сравнения элемент, соответствующий "меньшей" строке, перемещается в начало массива.

Недостаток такого подхода очевиден — числовые значения сортируются некорректно. Так, числа 1234 и 12345 будут признаны равными, т. к. первые четыре их символа совпадают. Так что простейшая сортировка поможет нам только в случае массивов, содержащих одни строки. В остальных же случаях нам придется задавать *Функцию сортировки*.

Функция, которая должна выполнять сортировку и которую мы передадим методу `sort`, должна принимать два параметра — два сравниваемых элемента массива. Возвращать же она должна одно из трех перечисленных далее значений:

- любое отрицательное число, если первый элемент массива должен находиться перед вторым (первый элемент "меньше" второго);
- ноль, если порядок следования элементов массива не должен измениться (оба элемента "равны");
- любое положительное число, если первый элемент массива должен находиться после второго (первый элемент "больше" второго).

> ***Народ советует***
>
> Часто функции сортировки возвращают значения –1, 0 и 1.

А уж в теле функции мы можем задать любой алгоритм сравнения элементов массива, который нам нужен. Мы даже сможем сортировать массив, каждый элемент которого представляет собой массив (кстати, это как раз наш случай) или экземпляр объекта.

Осталось только сказать, что метод `sort` объекта `Array` возвращает отсортированный массив и при этом изменяет массив, для которого он вызван.

Пример 1

Давайте возьмем Web-страницу из первого примера, описывающего использование объекта `DynamicTable`, и исправим ее код так, чтобы таблица выводилась уже отсортированной по столбцу **Категория**. Для этого нам понадобится добавить в формирующий таблицу Web-сценарий всего несколько строк (в приведенном далее фрагменте кода они выделены полужирным шрифтом):

```
. . .
tableBody[5] = new Array(JSPS_COMPILING, "Delphi");
tableBody[6] = new Array(JSPS_COMPILING, "Visual Basic");

function sortArray(pElement1, pElement2)
  {
    if (pElement1[0] < pElement2[0])
      return -1
    else
      if (pElement1[0] == pElement2[0])
        {
          if (pElement1[1] < pElement2[1])
            return -1
          else
            if (pElement1[1] == pElement2[1])
              return 0
            else
              return 1;
        }
      else
        return 1;
  }

tableBody.sort(sortArray);
var tableObject = new DynamicTable(document.body);
tableObject.createTable(tableBody, tableHead, null,
↳"Языки программирования");
tableObject.showTable();
. . .
```

Здесь мы сортируем массив сначала по первому элементу вложенного массива, а потом по его второму элементу.

Пример 2

Пусть посетитель нашей страницы сам выбирает, по какому столбцу таблицы выполнять сортировку. HTML-код страницы, предоставляющей такую возможность, приведен далее.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="dynamictable.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта DynamicTable помещено в
    файл сценария dynamictable.js -->
  </HEAD>
```

```
<BODY>
  <DIV ID="tableHolder"></DIV>
  <SCRIPT TYPE="text/javascript">
    var JSPS_COMPILING = "Компилируемый";
    var JSPS_INTERPRETING = "Интерпретируемый";
    var tableHead = new Array();
    tableHead[0] = new Array("Категория", "Язык");
    var tableBody = new Array();
    tableBody[0] = new Array(JSPS_COMPILING, "C++");
    tableBody[1] = new Array(JSPS_INTERPRETING, "JavaScript");
    tableBody[2] = new Array(JSPS_COMPILING, "Pascal");
    tableBody[3] = new Array(JSPS_COMPILING, "Java");
    tableBody[4] = new Array(JSPS_INTERPRETING, "Perl");
    tableBody[5] = new Array(JSPS_COMPILING, "Delphi");
    tableBody[6] = new Array(JSPS_COMPILING, "Visual Basic");
    var tableObject = new DynamicTable(document.all["tableHolder"]);

    function sortByCategory()
      {

        function sortArrayByCategory(pElement1, pElement2)
          {
            if (pElement1[0] < pElement2[0])
              return -1
            else
              if (pElement1[0] == pElement2[0])
                {
                  if (pElement1[1] < pElement2[1])
                    return -1
                  else
                    if (pElement1[1] == pElement2[1])
                      return 0
                    else
                      return 1;
                }
              else
                return 1;
          }

        tableBody.sort(sortArrayByCategory);
        tableObject.createTable(tableBody, tableHead, null,
        ↳"Языки программирования");
        tableObject.showTable();
      }
```

```
      function sortByLanguage()
        {

          function sortArrayByLanguage(pElement1, pElement2)
            {
              if (pElement1[1] < pElement2[1])
                return -1
              else
                if (pElement1[1] == pElement2[1])
                  return 0
                else
                  return 1;
            }

          tableBody.sort(sortArrayByLanguage);
          tableObject.createTable(tableBody, tableHead, null,
          ↳"Языки программирования");
          tableObject.showTable();
        }

      sortByCategory();
    </SCRIPT>
    <FORM>
      <INPUT TYPE="button" VALUE="По категории"
      ↳ONCLICK="sortByCategory();">
      <INPUT TYPE="button" VALUE="По названию языка"
      ↳ONCLICK="sortByLanguage();">
    </FORM>
  </BODY>
</HTML>
```

Как выполнить поиск в таблице?

Проблема

Мне необходимо выполнить поиск в таблице нужной строки. Как это сделать?

Решение

Самое простое — просмотреть все ячейки таблицы с помощью свойств DOM. Найдя ячейку, содержащую заданную строку, ее нужно как-то выделить, например, задать для нее цвет фона, отличный от цвета фона других ячеек.

Для выполнения поиска автор написал универсальную функцию `jspsSearchCell` (листинг 8.2). Формат ее вызова таков:

```
jspsSearchCell(<Таблица>, <Искомая строка>, <Стилевой класс выделения>
↳[, <Столбцы, в которых будет производиться поиск>]);
```

Первым параметром функции передается таблица, в которой нужно выполнить поиск, вторым — искомая строка. Третий параметр задает имя стилевого класса, который будет привязан к ячейкам, содержащим *Искомую строку*; это имя должно быть задано в строковом формате.

> ***Народ советует***
>
> Если нужно задать для ячеек таблицы, в которой будет производиться поиск с помощью функции `jspsSearchCell`, какой-либо стиль, можно использовать комбинированные стили. Например, так может выглядеть стиль для всей таблицы:
>
> ```
> .sometable { <Определение стиля> }
> ```
>
> А так — стиль для ячеек этой таблицы:
>
> ```
> .sometable TD { <Определение стиля> }
> ```
>
> После этого мы можем использовать функцию `jspsSearchCell` со спокойной совестью.

Третий, необязательный, параметр задает список номеров столбцов, в которых должен производиться поиск. Список должен представлять собой массив, содержащий эти номера. Если он пропущен, поиск будет вестись по всем столбцам таблицы.

Функция `jspsSearchCell` возвращает массив экземпляров объектов, соответствующих найденным ячейкам.

> ***Внимание!***
>
> Функция `jspsSearchCell` выполняет поиск только в секции тела таблицы. Автор посчитал, что в секции "поддона" и уж тем более в секции "шапки" искать смысла не имеет. Также эта функция выполняет поиск только в ячейках, содержащих один лишь текст, без тегов HTML.

Листинг 8.2. Функция `jspsSearchCell`, выполняющая поиск в таблице

```
function jspsSearchCell(pTable, pSubstring, pClassName, pColumnNumbers)
  {
    var columnNumbers = (pColumnNumbers) ? pColumnNumbers.join(" ") : "";
    pSubstring = pSubstring.toLowerCase();
    var resultCells = new Array();
    var bodySectionObject = pTable.getElementsByTagName("TBODY");
    bodySectionObject = bodySectionObject[0];
    for (var i = 0; i < bodySectionObject.childNodes.length; i++)
```

```
    {
      var rowObject = bodySectionObject.childNodes[i];
      for (var j = 0; j < rowObject.childNodes.length; j++)
        if ((columnNumbers == "") ||
        ↳(columnNumbers.indexOf(j.toString()) != -1))
          {
            var cellObject = rowObject.childNodes[j];
            cellObject.className = "";
            var cellText = cellObject.firstChild.nodeValue;
            if (cellText)
              {
                cellText = cellText.toLowerCase();
                if (cellText.indexOf(pSubstring) != -1)
                  {
                    cellObject.className = pClassName;
                    resultCells[resultCells.length] = cellObject;
                  }
              }
          }
    }
  return resultCells;
}
```

Кстати, если мы собираемся пользоваться для поиска по таблице функцией `jspsSearchCell`, нам также пригодится функция `jspsClearSearchCell` (листинг 8.3). Она занимается тем, что удаляет со всех ячеек выделение, установленное функцией `jspsSearchCell`. Формат ее вызова очень прост:

```
jspsClearSearchCell(<Таблица>);
```

Единственным параметром этой функции передается таблица, с ячеек которой нужно снять выделение. Значения эта функция не возвращает.

Листинг 8.3. Функция `jspsClearSearchCell`, снимающая выделение ячеек таблицы, установленное функцией `jspsSearchCell`

```
function jspsClearSearchCell(pTable)
  {
    var bodySectionObject = pTable.getElementsByTagName("TBODY");
    bodySectionObject = bodySectionObject[0];
    for (var i = 0; i < bodySectionObject.childNodes.length; i++)
      {
        var rowObject = bodySectionObject.childNodes[i];
        for (var j = 0; j < rowObject.childNodes.length; j++)
```

```
            rowObject.childNodes[j].className = "";
        }
    }
```

Хорошая идея!

Добавьте объявление этих функций в файл сценариев dynamictable.js, содержащий объявление объекта `DynamicTable`. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="dynamictable.js"></SCRIPT>
```

Пример

Далее приведен HTML-код страницы, выводящей на экран таблицу с возможностью поиска.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="dynamictable.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта DynamicTable и функций
    jspsSearchCell и jspsClearSearchCell помещено в файл сценария
    dynamictable.js -->
    <STYLE>
      .found { border: thin solid #CCCCCC; }
    </STYLE>
  </HEAD>
  <BODY>
    <DIV ID="tableHolder"></DIV>
    <SCRIPT TYPE="text/javascript">
      var JSPS_COMPILING = "Компилируемый";
      var JSPS_INTERPRETING = "Интерпретируемый";
      var tableHead = new Array();
      tableHead[0] = new Array("Категория", "Язык");
      var tableBody = new Array();
      tableBody[0] = new Array(JSPS_COMPILING, "C++");
      tableBody[1] = new Array(JSPS_INTERPRETING, "JavaScript");
      tableBody[2] = new Array(JSPS_COMPILING, "Pascal");
      tableBody[3] = new Array(JSPS_COMPILING, "Java");
      tableBody[4] = new Array(JSPS_INTERPRETING, "Perl");
      tableBody[5] = new Array(JSPS_COMPILING, "Delphi");
      tableBody[6] = new Array(JSPS_COMPILING, "Visual Basic");
      var tableObject = new DynamicTable(document.all["tableHolder"]);
      tableObject.createTable(tableBody, tableHead, null,
      ↳"Языки программирования");
      tableObject.showTable();
```

```
      function searchCell()
        {
          var txtSearchObject = document.all["txtSearch"];
          var txtSubstring = txtSearchObject.value;
          if (txtSubstring)
            var foundCells = jspsSearchCell(tableObject.table,
            ↳txtSubstring, "found");
        }
    </SCRIPT>
    <FORM>
      <INPUT TYPE="text" ID="txtSearch">
      <INPUT TYPE="button" VALUE="Найти" ONCLICK="searchCell();">
      <INPUT TYPE="button" VALUE="Убрать выделение"
      ↳ONCLICK="jspsClearSearchCell(tableObject.table);">
    </FORM>
  </BODY>
</HTML>
```

Народ предупреждает!

В случае таблиц, выводимых постранично, нам придется вызывать функцию `jspsSearchCell` сразу же после вывода на экран очередной "страницы" таблицы.

Как отфильтровать данные в таблице?

Проблема

Мне нужно выполнить фильтрацию таблицы, т. е. вывести в нее только те данные, что удовлетворяют определенному критерию. Можно ли это сделать?

Решение 1

Очевиднее некуда — выбрать из массива, на основе которого создается таблица, данные, удовлетворяющие заданному критерию, поместить их в другой массив и на основе уже этого массива создать таблицу.

Народ советует

Если набор критериев фильтрации ограничен, проще всего создать несколько массивов — по одному на каждый критерий. При выборе критерия фильтрации пользователем нам будет достаточно найти соответствующий массив и использовать его для создания таблицы.

Пример

Далее приведен HTML-код страницы, выводящей на экран таблицу — список языков программирования. При этом посетитель имеет возможность выбрать, какие языки отображать — интерпретируемые или компилируемые.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="dynamictable.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта DynamicTable помещено в
    файл сценария dynamictable.js -->
  </HEAD>
  <BODY>
    <DIV ID="tableHolder"></DIV>
    <SCRIPT TYPE="text/javascript">
      var JSPS_COMPILING = "Компилируемый";
      var JSPS_INTERPRETING = "Интерпретируемый";
      var tableHead = new Array();
      tableHead[0] = new Array("Категория", "Язык");
      var tableBodyAll = new Array();
      tableBodyAll[0] = new Array(JSPS_COMPILING, "C++");
      tableBodyAll[1] = new Array(JSPS_INTERPRETING, "JavaScript");
      tableBodyAll[2] = new Array(JSPS_COMPILING, "Pascal");
      tableBodyAll[3] = new Array(JSPS_COMPILING, "Java");
      tableBodyAll[4] = new Array(JSPS_INTERPRETING, "Perl");
      tableBodyAll[5] = new Array(JSPS_COMPILING, "Delphi");
      tableBodyAll[6] = new Array(JSPS_COMPILING, "Visual Basic");
      // Объявляем два массива — для интерпретируемых и компилируемых
      // языков программирования соответственно
      var tableBodyI = new Array();
      var tableBodyC = new Array();
      // Заносим интерпретируемые и компилируемые языки в соответствующие
      // им массивы
      for (var i = 0; i < tableBodyAll.length; i++)
        if (tableBodyAll[i][0] == JSPS_INTERPRETING)
          tableBodyI[tableBodyI.length] = tableBodyAll[i]
        else
          tableBodyC[tableBodyC.length] = tableBodyAll[i];
      var tableObject = new DynamicTable(document.all["tableHolder"]);

      // Функции для создания таблиц на основе различных массивов
      function showC()
        {
          tableObject.createTable(tableBodyC, tableHead, null,
          ↳"Языки программирования");
          tableObject.showTable();
        }
```

```
      function showI()
        {
          tableObject.createTable(tableBodyI, tableHead, null,
          ↳"Языки программирования");
          tableObject.showTable();
        }

      function showAll()
        {
          tableObject.createTable(tableBodyAll, tableHead, null,
          ↳"Языки программирования");
          tableObject.showTable();
        }

      showAll();
    </SCRIPT>
    <FORM>
      <INPUT TYPE="button" VALUE="Вывести компилируемые"
      ↳ONCLICK="showC();">
      <INPUT TYPE="button" VALUE="Вывести интерпретируемые"
      ↳ONCLICK="showI();">
      <INPUT TYPE="button" VALUE="Вывести все" ONCLICK="showAll();">
    </FORM>
  </BODY>
</HTML>
```

Решение 2

Также простое — просмотреть все ячейки таблицы с помощью свойств DOM. Все строки, не удовлетворяющие заданному критерию фильтрации, нужно скрыть, присвоив свойству `display` вложенного объекта `style` строку `"none"`. А все строки, удовлетворяющие критерию фильтрации, нужно сделать видимыми, для чего свойству `display` вложенного объекта `style` нужно присвоить строку `"table-row"`.

> ***Народ предупреждает!***
>
> В этом случае нужно использовать именно свойство `display`, а не `visibility`. Как мы знаем из *главы 5*, свойство `display` позволяет скрыть элемент страницы полностью, словно он и не был определен в коде HTML. А свойство `visibility` только делает элемент страницы невидимым, но оставляет его на странице, где он виден как пустое пространство.
>
> Internet Explorer здесь снова демонстрирует свои, скажем так, особенности реализации. Он не поддерживает значение `"table-row"` свойства `display` (об этом говорилось в той же *главе 5*), а ведь именно это значение позволяет ото-

бразить элемент страницы как строку таблицы. Поэтому в случае Internet Explorer приходится использовать значение "block" этого свойства как наиболее близкое к "table-row" по результату действия.

Для выполнения такого рода фильтрации автор написал универсальную функцию jspsFilterTable (листинг 8.4). Вот формат ее вызова:

```
jspsFilterTable(<Таблица>, <Функция фильтрации>
↳[, <Дополнительный параметр функции фильтрации>]);
```

Первым параметром функции передается таблица, которую нужно отфильтровать. Здесь все ясно.

Второй же параметр задает функцию фильтрации, проверяющую, удовлетворяют ли данные, помещенные в ячейках строки, заданному критерию. Эта функция должна принимать единственный обязательный параметр — строку таблицы — и возвращать значение true, если данные в ячейках этой строки удовлетворяют критерию фильтрации, и false в противном случае. Также функция фильтрации может принимать *Дополнительный параметр*, который передается функции jspsFilterTable.

Функция jspsFilterTable не возвращает значения.

Внимание!

Функция jspsFilterTable выполняет фильтрацию только секции тела таблицы. Автор посчитал, что секцию "поддона" и уж тем более секцию "шапки" фильтровать смысла не имеет.

Листинг 8.4. Функция jspsFilterTable, выполняющая фильтрацию таблицы

```
function jspsFilterTable(pTable, pFilterFunction, pAdditional)
  {
    var piObject = jspsGetProgramInfo();
    var isIE = (piObject.programName == JSPS_GPI_MSIE)
    var bodySectionObject = pTable.getElementsByTagName("TBODY");
    bodySectionObject = bodySectionObject[0];
    for (var i = 0; i < bodySectionObject.childNodes.length; i++)
      {
        var rowObject = bodySectionObject.childNodes[i];
        rowObject.style.display = (pFilterFunction(rowObject,
        ↳pAdditional)) ? ((isIE) ? "block" : "table-row") : "none";
      }
  }
```

Если мы собираемся пользоваться для фильтрации таблицы функцией jspsFilterTable, нам также пригодится функция jspsClearFilterTable (лис-

тинг 8.5). Она делает все ячейки таблицы видимыми, аннулируя результат вызова функции jspsFilterTable. Функция имеет очень простой формат вызова:

```
jspsClearFilterTable(<Таблица>);
```

Единственным параметром этой функции передается таблица, ячейки которой нужно сделать видимыми. Значения эта функция не возвращает.

Листинг 8.5. Функция jspsClearFilterTable, аннулирующая результат выполнения функции jspsFilterTable, делая все ячейки таблицы видимыми

```
function jspsClearFilterTable(pTable)
  {
    var piObject = jspsGetProgramInfo();
    var isIE = (piObject.programName == JSPS_GPI_MSIE)
    var bodySectionObject = pTable.getElementsByTagName("TBODY");
    bodySectionObject = bodySectionObject[0];
    for (var i = 0; i < bodySectionObject.childNodes.length; i++)
      bodySectionObject.childNodes[i].style.display = (isIE) ? "block" :
      ↳"table-row";
  }
```

Внимание!

Объявления функций jspsFilterTable и jspsClearFilterTable используют функцию jspsGetProgramInfo, чье объявление приведено в листинге 3.1.

Хорошая идея!

Добавьте объявление этих функций в файл сценариев dynamictable.js, содержащий объявление объекта DynamicTable. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="dynamictable.js"></SCRIPT>
```

Пример

Что ж, напишем еще одну страницу, которая будет выводить на экран таблицу — список языков программирования с возможностью фильтрации их по категории.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT SRC="browserdetect.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsGetProgramInfo
    помещено в файл сценария browserdetect.js -->
```

```
  <SCRIPT SRC="dynamictable.js"></SCRIPT>
  <!-- Предполагается, что объявление объекта DynamicTable и функций
  jspsFilterTable и jspsClearFilterTable помещено в файл сценария
  dynamictable.js -->
</HEAD>
<BODY>
  <DIV ID="tableHolder"></DIV>
  <SCRIPT TYPE="text/javascript">
    var JSPS_COMPILING = "Компилируемый";
    var JSPS_INTERPRETING = "Интерпретируемый";
    var tableHead = new Array();
    tableHead[0] = new Array("Категория", "Язык");
    var tableBody = new Array();
    tableBody[0] = new Array(JSPS_COMPILING, "C++");
    tableBody[1] = new Array(JSPS_INTERPRETING, "JavaScript");
    tableBody[2] = new Array(JSPS_COMPILING, "Pascal");
    tableBody[3] = new Array(JSPS_COMPILING, "Java");
    tableBody[4] = new Array(JSPS_INTERPRETING, "Perl");
    tableBody[5] = new Array(JSPS_COMPILING, "Delphi");
    tableBody[6] = new Array(JSPS_COMPILING, "Visual Basic");
    var tableObject = new DynamicTable(document.all["tableHolder"]);
    tableObject.createTable(tableBody, tableHead, null,
    ↳"Языки программирования");
    tableObject.showTable();

    // Функция фильтрации. Здесь мы передаем значение, с которым должен
    // сравниваться содержащийся в ячейке текст, дополнительным
    // параметром
    function filterTable(pRowObject, pAdditional)
      {
        return (pRowObject.childNodes[0].firstChild.nodeValue ==
        ↳pAdditional);
      }
  </SCRIPT>
  <FORM>
    <INPUT TYPE="button" VALUE="Вывести компилируемые"
    ↳ONCLICK="jspsFilterTable(tableObject.table, filterTable,
    ↳JSPS_COMPILING);">
    <INPUT TYPE="button" VALUE="Вывести интерпретируемые"
    ↳ONCLICK="jspsFilterTable(tableObject.table, filterTable,
    ↳JSPS_INTERPRETING);">
    <INPUT TYPE="button" VALUE="Вывести все"
    ↳ONCLICK="jspsClearFilterTable(tableObject.table);">
```

```
    </FORM>
  </BODY>
</HTML>
```

Как видим, код страницы получился много компактнее, чем в предыдущем примере. Вот что значит использование файлов сценариев!

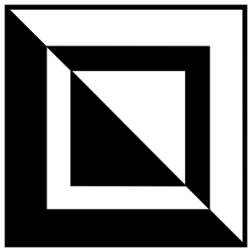
Народ предупреждает!

В случае таблиц, выводимых постранично, нам придется вызывать функцию `jspsFilterTable` сразу же после вывода на экран очередной "страницы" таблицы.

Что дальше?

Длинный получился урок... Но зато сколько мы всего узнали! Мы научились создавать таблицы программно, сортировать их, фильтровать и искать в них нужные нам данные. И учитель-народ облегченно ставит нам пятерку...

Следующее занятие будет посвящено фреймам. Звенит звонок!.. Начинаем!



ГЛАВА 9

Работа с фреймами

Фреймы и фреймовый дизайн Web-страниц давно уже вышли из моды, но все еще кое-где встречаются. Поэтому работать с ними народу приходится довольно часто. Посмотрим, что же народ с ними творит, и поучимся у него.

Как получить доступ к нужному фрейму и его содержимому?

Проблема

Мне нужно из Web-сценария, помещенного в открытую во фрейме страницу, получить доступ к другому фрейму того же набора. Как это сделать?

Решение

Если из страницы, загруженной в один из фреймов, нужно получить доступ к другому фрейму того же набора, следует использовать два свойства объекта `window`:

- `parent` — предоставляет доступ к окну Web-обозревателя, в котором загружена страница набора фреймов;
- `top` — предоставляет доступ к окну Web-обозревателя, в котором загружена страница самого верхнего набора фреймов в иерархии.

Понятно, что оба этих свойства возвращают экземпляр объекта `window` — соответствующее окно Web-обозревателя.

Народ замечает

Объект `window` поддерживает также свойство `self`, предоставляющее доступ к экземпляру объекта `window` — текущему фрейму. Только непонятно, зачем оно нужно.

Итак, окно, в котором открыт набор фреймов, мы получили. Как теперь добраться до нужного фрейма? Воспользоваться коллекцией `frames` объекта `window`, которая содержит все фреймы данного набора. Для доступа к нужному фрейму мы можем использовать как численный индекс, так и буквенное имя, заданное атрибутом `NAME` тега `<FRAME>`.

> ***Народ предупреждает!***
>
> Имена фреймов следует задавать только с помощью атрибута `NAME` тега `<FRAME>`, который, хоть и объявлен устаревшим и не рекомендован к использованию, но поддерживается всеми Web-обозревателями. Что касается стандартного атрибута `ID`, то для тега `<FRAME>` он почему-то не поддерживается. Впрочем, фреймы до сих пор толком не стандартизованы...

```
var frmTitleObject = parent.frames["title"];
```

Это выражение поместит в переменную `frmTitleObject` экземпляр объекта, соответствующий фрейму с именем `title`.

Ну вот, получили мы фрейм. И что теперь с ним делать? Да что угодно! Ведь фрейм — это экземпляр объекта `window`, своего рода "окошко" в окне Web-обозревателя. А значит, мы можем обратиться к свойству `document` этого объекта и получить доступ к Web-странице, открытой в данном фрейме.

```
frmTitleObject.document.write("<H1>Мой сайт</H1>");
```

А это выражение поместит во фрейм `title` заголовок с текстом "Мой сайт".

Свойства `parent` и `top` имеют одно различие, о котором мы сейчас поговорим. Предположим, что мы создали страницу frameset1.html, содержащую набор из двух фреймов `frame1` и `frame2`. Во фрейм `frame1` мы загрузили обычную Web-страницу page1.html, а во фрейм `frame2` — страницу frameset2.html, содержащую другой набор фреймов. Последний содержит фреймы `frame3` (в него загружена страница page3.html) и `frame4` (страница page4.html).

Теперь, если мы обратимся к свойству `parent` из страницы page1.html (фрейм `frame1` "внешнего" набора frameset1.html), то получим экземпляр объекта `window`, соответствующий окну Web-обозревателя, в который загружен "внешний" набор фреймов frameset1.html. То же самое мы получим, если обратимся к свойству `top` из этой же страницы. Видно, что в этом случае свойства `parent` и `top` выполняют одинаковую функцию.

Совсем другая картина появится, если мы попытаемся обратиться к этим свойствам из Web-сценария, помещенного в страницу page4.html (фрейм `frame4` "внутреннего" набора frameset2.html). Здесь свойство `parent` вернет экземпляр объекта `window`, соответствующий фрейму `frame2` "внешнего" набора frameset1.html, в который загружен "внутренний" набор frameset2.html.

Свойство `top` же вернет окно Web-обозревателя, в котором загружен "внешний" набор frameset1.html.

Народ замечает

Вообще, отдельная страница набора фреймов, открытая во фрейме другого набора, — не очень хороший стиль Web-дизайна. Хотя иногда такая ситуация встречается. Например, некоторые бесплатные хостинг-провайдеры открывают все страницы пользователя во фрейме (другие фреймы могут быть заняты рекламой), и если сайт пользователя построен на основе фреймов, мы получим как раз такой случай.

Пример

Давайте создадим набор из двух фреймов. В первом фрейме поместим страницу с двумя кнопками, задающими цвет текста страницы во втором фрейме.

Вот HTML-код страницы jspsSimpleFrameSet.html, содержащей сам набор фреймов:

```
<HTML>
  <HEAD>
    <TITLE>Набор фреймов</TITLE>
  </HEAD>
  <FRAMESET COLS="50%, 50%">
    <FRAME SRC="jspsSimpleFrameLeft.html" NAME="left">
    <FRAME SRC="jspsSimpleFrameRight.html" NAME="right">
  </FRAMESET>
</HTML>
```

Вот код страницы jspsSimpleFrameLeft.html, содержащей кнопки:

```
<HTML>
  <HEAD>
    <TITLE>Левый фрейм</TITLE>
  </HEAD>
  <BODY>
    <FORM>
      <INPUT TYPE="button" VALUE="Красный"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#FF0000';">
      <INPUT TYPE="button" VALUE="Зеленый"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#00FF00';">
      <INPUT TYPE="button" VALUE="Синий"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#0000FF';">
      <INPUT TYPE="button" VALUE="Черный"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#000000';">
    </FORM>
```

```
    <SCRIPT TYPE="text/javascript">
      var frmRightObject = parent.frames["right"];
    </SCRIPT>
  </BODY>
</HTML>
```

А код страницы jspsSimpleFrameRight.html, содержащей текст, совсем прост:

```
<HTML>
  <HEAD>
    <TITLE>Правый фрейм</TITLE>
  </HEAD>
  <BODY>
    <P>Это содержимое фрейма.</P>
  </BODY>
</HTML>
```

Загрузим в Web-обозревателе страницу jspsSimpleFrameSet.html и попробуем понажимать на кнопки. Должно работать.

Как проверить, открыта ли данная страница во фрейме или нет?

Проблема

Мне нужно проверить, открыта ли моя Web-страница во фрейме или же в целом окне Web-обозревателя. Как это сделать?

Решение

Здесь нет ничего сложного. Достаточно проверить, равно ли свойство `length` коллекции `frames` объекта `window` нулю. (Это свойство, как мы знаем, возвращает количество элементов в коллекции.) Если это так, значит, фреймов на данной странице не существует, т. е. она загружена непосредственно в окно Web-обозревателя.

Пример 1

```
if (parent.frames.length == 0)
  document.write("Здесь нет фреймов")
else
  document.write("Здесь присутствуют " + parent.frames.length.toString()
  ↳+ " фреймов");
```

Этот небольшой сценарий выведет на страницу текст "Здесь нет фреймов" или "Здесь присутствуют <*Количество фреймов*> фреймов".

Пример 2

Давайте сделаем действительно полезную вещь. И заодно решим одну весьма актуальную проблему.

Если зайти на одну из страниц сайта, построенного на основе фреймов, по прямой ссылке (вида **http://www.somesite.ru/somepage.html**), эта страница откроется прямо в окне Web-обозревателя, без фреймов. В результате посетитель не сможем перейти на другие страницы этого сайта, т. к. набор гиперссылок для перехода на эти страницы всегда помещается в другие фреймы, которые в данный момент не выводятся на экран.

Давайте же напишем небольшой сценарий, который в случае открытия страницы непосредственно в окне Web-обозревателя поместит на нее ссылку на главную страницу сайта, содержащую набор фреймов. Так посетитель сможет перейти, по крайней мере, на главную страницу и оттуда без проблем продолжить "путешествие" по сайту.

Далее приведен HTML-код страницы, содержащей такой сценарий.

```
<HTML>
  <HEAD>
    <TITLE>Фрейм</TITLE>
  </HEAD>
  <BODY>
    <P>Это содержимое фрейма.</P>
    <SCRIPT TYPE="text/javascript">
      var hrefToHomePage = "\default.html";
      if (parent.frames.length == 0)
        document.write("<P><A HREF=\"" + hrefToHomePage +
        ↳"\">Перейти на главную страницу</A></P>");
    </SCRIPT>
  </BODY>
</HTML>
```

Народ советует

Следует всегда на страницах сайта, построенного на основе фреймов, предусматривать нечто подобное. Иначе посетитель, попавший на одну из его страниц по прямой ссылке (например, с поисковой машины), не сможет добраться до других страниц такого сайта.

Как принудительно загрузить главную страницу сайта во фрейме?

Проблема

Автор описал, как выяснить, загружена ли страница во фрейме или непосредственно в окне Web-обозревателя. И даже рассказал, как поместить на стра-

ницу гиперссылку, позволяющую перейти на главную страницу сайта. Но если по прямой ссылке открыта сама главная страница (в смысле, страница с изначальным содержимым основного фрейма), разумнее будет сразу же переоткрыть ее, но уже во фрейме. Не так ли?

Решение

Действительно, так... Ну что ж, автор книги сейчас исправит свою ошибку.

Вот формат вызова функции `jspsReloadInFrameset` (листинг 9.1), проверяющей, загружена ли текущая Web-страница во фрейме, и, если это не так, выполняющей переход на главную страницу сайта:

```
jspsReloadInFrameset(<Интернет-адрес главной страницы сайта>);
```

Единственный параметр этой функции задает интернет-адрес главной страницы. Понятно, что он указывается в строковом виде. Функция не возвращает значения.

Листинг 9.1. Функция `jspsReloadInFrameset`, проверяющая, загружена ли текущая Web-страница во фрейме, и, если это не так, выполняющая переход на главную страницу сайта

```
function jspsReloadInFrameset(pHref)
  {
    if (parent.frames.length == 0)
      window.location.href = pHref;
  }
```

Хорошая идея!

Поместите объявление этой функции в файл сценариев frames.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="frames.js"></SCRIPT>
```

Пример

Немного исправим код созданной нами ранее Web-страницы jspsSimpleFrameLeft.html, введя проверку, открыта ли она во фрейме.

```
<HTML>
  <HEAD>
    <TITLE>Левый фрейм</TITLE>
    <SCRIPT SRC="draw.js"></SCRIPT>
    <!-- Предполагаем, что объявление функции jspsReloadInFrameset
    находится в файле сценариев frames.js -->
    <SCRIPT TYPE="text/javascript">
```

```
      jspsReloadInFrameset("jspsSimpleFrameSet.html");
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM>
      <INPUT TYPE="button" VALUE="Красный"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#FF0000';">
      <INPUT TYPE="button" VALUE="Зеленый"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#00FF00';">
      <INPUT TYPE="button" VALUE="Синий"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#0000FF';">
      <INPUT TYPE="button" VALUE="Черный"
      ↳ONCLICK="frmRightObject.document.body.style.color = '#000000';">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      var frmRightObject = parent.frames["right"];
    </SCRIPT>
  </BODY>
</HTML>
```

Аналогично можно добавить такую же функциональность и в страницу jspsSimpleFrameRight.html.

Народ замечает

В большинстве случаев такую функциональность стоит добавить только в главную страницу сайта, т. е. в страницу с изначальным содержимым основного фрейма (того, в котором выводится основное содержимое сайта). Остальным страницам сайта она ни к чему. Посетитель, даже если он пришел на страницу по прямой ссылке, должен получить именно эту страницу.

Как при щелчке на гиперссылке обновить содержимое сразу нескольких фреймов?

Проблема

Я создал сложный набор фреймов и теперь не могу придумать, как обновить при щелчке на одной гиперссылке содержимое сразу нескольких фреймов.

Решение

Ну, о подобных штуках мы упоминали еще в *главе 7*, посвященной средствам навигации. Мы писали функцию-обработчик, выполняющую загрузку нужной страницы, и привязывали ее к событию `onClick` гиперссылки. Обновле-

ние содержимого нескольких фреймов можно также выполнить с помощью подобной функции-обработчика.

Пример

Далее приведен фрагмент HTML-кода Web-страницы с набором гиперссылок, загруженной в один из фреймов набора. При щелчке на гиперссылке сразу в два соседних фрейма (`frm1` и `frm2`) загружаются новые Web-страницы.

```
<HEAD>
  . . .
  <SCRIPT TYPE="text/javascript">
    function loadPages()
      {
        var frm1Object = parent.frames["frm1"];
        var frm2Object = parent.frames["frm2"];
        frm1Object.location.href = "page1.html";
        frm2Object.location.href = "page2.html";
      }
  </SCRIPT>
  . . .
</HEAD>
<BODY>
  . . .
  <P><A HREF="#" ONCLICK="loadPages();">Обновить фреймы</A></P>
  . . .
</BODY>
```

Как изменить текст в заголовке окна Web-обозревателя при загрузке новой страницы во фрейм?

Проблема

Не я один с ней сталкивался. Обычно в заголовке окна Web-обозревателя отображается заголовок Web-страницы, заданный тегом `<TITLE>` в ее секции заголовка (в теге `<HEAD>`). Но в случае фреймового Web-дизайна там постоянно отображается заголовок страницы с набором фреймов. А я хочу выводить туда заголовки загружаемых в основном фрейме страниц.

Решение

Чтобы задать произвольный текст для заголовка окна Web-обозревателя, достаточно присвоить его свойству `title` объекта `document`. Если же нужно изме-

нить текст заголовка окна, в котором открыт набор фреймов, из страницы, открытой в одном из фреймов этого набора, сначала нужно получить доступ к этому окну через свойства `parent` или `top`. Например:

```
parent.document.title = "Новости сайта";
```

После выполнения этого выражения в заголовке окна Web-обозревателя появится текст "Новости сайта".

Также можно воспользоваться универсальной функцией `jspsSetTitle` (листинг 9.2), которая не принимает параметров и не возвращает результата. Эта функция помещает в заголовок окна Web-обозревателя текст заголовка текущей страницы.

Листинг 9.2. Функция `jspsSetTitle`, помещающая в заголовок окна Web-обозревателя текст заголовка текущей Web-страницы

```
function jspsSetTitle()
  {
    parent.document.title = document.title;
  }
```

Хорошая идея!

Поместите объявление этой функции в файл сценариев frames.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

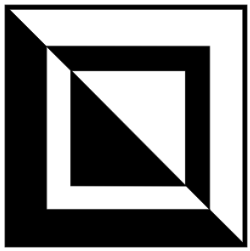
```
<SCRIPT SRC="frames.js"></SCRIPT>
```

Думается, пример использования этой функции приводить не стоит.

Что дальше?

Разговор о фреймах получился довольно коротким. В самом деле, что там может быть сложного?.. И что вообще эти фреймы позволяют нам сделать?..

А вот свободно позиционируемые элементы позволяют сделать многое. Например, создать настоящую анимацию на Web-странице. Или сделать графический курсор мыши. Этим мы займемся в следующей главе.



ГЛАВА 10

Управление свободно позиционируемыми элементами

Свободно позиционируемые элементы (их также кратко называют *свободными элементами*) могут располагаться на Web-странице как угодно, иметь какие угодно размеры и даже перекрывать друг друга. Они создаются с помощью тегов <DIV> (так называемые *свободно позиционируемые контейнеры*), <IMG> (*свободно позиционируемые изображения*) и некоторых других с помощью особых атрибутов стилей CSS, которые, собственно, и задают их местоположение и размеры.

Но самое замечательное в свободных контейнерах то, что их местоположением и размерами мы можем управлять из Web-сценариев. Что это значит? Да хотя бы то, что мы можем с их помощью создавать настоящую анимацию — элементы страницы, движущиеся по любым, даже весьма сложным траекториям. С обычными (фиксированными) элементами страницы такой номер не пройдет.

Собственно, со свободно позиционируемыми элементами мы уже познакомились. В *главе 6* мы рисовали на Web-странице произвольные фигуры из точек, для размещения которых использовали свободно позиционируемые изображения (теги <IMG>). А в *главе 7* мы поражали посетителей своих страниц всплывающими подсказками — ведь это тоже были свободные элементы (а именно свободные контейнеры <DIV>)!

Так что можно сказать, что первое знакомство с этими замечательными элементами страницы у нас уже состоялось. Настала пора его продолжить.

Простейшие эффекты со свободными элементами

Начнем мы, как всегда, с самого простого. Мы научимся позиционировать и менять размеры свободных элементов программно, выравнивать их по краю

страницы, по краю другого элемента страницы и динамически менять местоположение свободного элемента при изменении размеров страницы.

Как управлять местоположением и размерами свободного элемента?

Проблема

Начитавшись книг Дронова, я создал страницу с множеством свободно позиционируемых элементов. Как мне теперь изменить их размеры и местоположение?

Решение

Нет ничего проще! Специально для этого объект `style`, поддерживаемый всеми объектами — элементами страницы, предоставляет четыре свойства:

- `left` — горизонтальная координата левой границы элемента;
- `top` — вертикальная координата верхней границы элемента;
- `width` — ширина элемента;
- `height` — высота элемента.

Кроме того, доступны свойства `display` и `visibility`, описанные в *главе 5*. С помощью этих свойств можно скрыть элемент, сделав его невидимым.

> ***Народ советует***
>
> В случае свободно позиционируемого элемента для его скрытия достаточно будет свойства `visibility` объекта `style`. Так, чтобы скрыть элемент, достаточно присвоить этому свойству значение `"hidden"`. Сделать элемент видимым можно, присвоив этому свойству значение `"visible"`.

Пример

Приведенный далее фрагмент HTML-кода создаст на странице свободно позиционируемый контейнер `cont`, поместит его в точку с координатами (200, 100), задаст для него ширину, равную 400 пикселам, и высоту, равную 50 пикселам.

```
<DIV ID="cont" STYLE="position: absolute">Это контейнер.</DIV>
<SCRIPT TYPE="text/javascript">
  var contObject = document.all["cont"];
  contObject.style.left = 200;
  contObject.style.top = 100;
  contObject.style.width = 400;
  contObject.style.height = 50;
</SCRIPT>
```

Как получить координаты и размеры элемента?

Проблема

Хорошо, координаты и размеры свободного элемента я задать могу. Но как их получить?

Решение

Специально для этого случая экземпляр объекта, соответствующего элементу страницы (не объект `style`!), поддерживает целый набор свойств, которые перечислены в табл. 10.1. Все они доступны только для чтения.

Таблица 10.1. *Свойства экземпляра объекта — элемента Web-страницы, позволяющие узнать его размеры*

Свойство	Описание
`clientHeight`	Возвращает высоту элемента страницы без учета рамок, отступов и полос прокрутки
`clientLeft`	Возвращает смещение левого края элемента страницы относительно левого края родителя в пикселах без учета рамок, отступов и полос прокрутки
`clientTop`	Возвращает смещение верхнего края элемента страницы относительно верхнего края родителя в пикселах без учета рамок, отступов и полос прокрутки
`clientWidth`	Возвращает ширину элемента страницы без учета рамок, отступов и полос прокрутки
`height`	Возвращает значение атрибута `HEIGHT` тега элемента страницы
`offsetHeight`	Возвращает высоту элемента страницы в пикселах
`offsetLeft`	Возвращает смещение левого края элемента страницы относительно левого края родителя в пикселах
`offsetTop`	Возвращает смещение верхнего края элемента страницы относительно верхнего края родителя в пикселах
`offsetWidth`	Возвращает ширину элемента страницы в пикселах относительно ширины родителя
`scrollHeight`	Возвращает полную высоту содержимого элемента страницы в пикселах (т. е. высоту, которую элемент страницы принял бы, чтобы вместить все свое содержимое). Имеет смысл только в том случае, если содержимое не помещается в элемент страницы, и в нем появилась вертикальная полоса прокрутки
`scrollLeft`	Положение горизонтальной полосы прокрутки элемента страницы в пикселах
`scrollTop`	Положение вертикальной полосы прокрутки элемента страницы в пикселах

Таблица 10.1 *(окончание)*

Свойство	Описание
scrollWidth	Возвращает полную ширину содержимого элемента страницы в пикселах (т. е. ширину, которую элемент страницы принял бы, чтобы вместить все свое содержимое). Имеет смысл только в том случае, если содержимое не помещается в элемент страницы, и в нем появилась горизонтальная полоса прокрутки
width	Возвращает значение атрибута HEIGHT тега элемента страницы

Народ замечает

Самое интересное в представленных в табл. 10.1 свойствах то, что они поддерживаются всеми элементами страницы, а не только свободными элементами. Мы, собственно, этим уже пользовались в *главе 7*, когда создавали всплывающие подсказки (см. листинг 7.1).

Как выровнять свободный элемент по краю его родителя?

Проблема

Мне нужно выровнять свободно позиционируемый элемент по определенному краю его родителя (Web-страницы или другого, внешнего, элемента). Как это сделать?

Решение

Использование универсальной функции jspsAlignContainer (листинг 10.1). Формат ее вызова таков:

```
jspsAlignContainer(<Свободный элемент>[, <Край родителя>
[, <Растягивать свободный элемент>]]]);
```

Первым параметром передается сам свободный элемент, который нужно выровнять по краю родителя. Это единственный обязательный параметр функции jspsAlignContainer.

Вторым параметром этой функции передается числовое значение, задающее край родителя, по которому нужно выровнять элемент. Вот все доступные значения этого параметра (приведены имена псевдоконстант, значения которых можно найти в листинге 10.1):

- JSPS_AC_LEFT — по левому краю;
- JSPS_AC_LEFTTOP — по левому и верхнему краю родителя (т. е. поместить элемент в его левый верхний угол);

- JSPS_AC_TOP — по верхнему краю;
- JSPS_AC_RIGHTTOP — по правому и верхнему краю (в правый верхний угол);
- JSPS_AC_RIGHT — по правому краю;
- JSPS_AC_RIGHTBOTTOM — по правому и нижнему краю (в правый нижний угол);
- JSPS_AC_BOTTOM — по нижнему краю;
- JSPS_AC_LEFTBOTTOM — по левому и нижнему краю (в левый нижний угол);
- JSPS_AC_CENTER — по центру;
- JSPS_AC_FULL — полное выравнивание (т. е. элемент будет растянут на всю "территорию" родителя).

Если второй параметр не указан, элемент будет выровнен по центру, как если бы мы указали значение JSPS_AC_CENTER.

Третий параметр функции jspsAlignContainer позволяет указать, растягивать ли выравниваемый элемент вдоль края родителя, по которому выполняется выравнивание, так, чтобы он занял его полностью. Если указано значение true, элемент будет растянут; значение false предписывает оставить размеры элемента неизменными. Пропуск третьего параметра эквивалентен указанию значения false.

Нужно иметь в виду, что третий параметр функции jspsAlignContainer имеет смысл только при значениях второго параметра, равных JSPS_AC_LEFT, JSPS_AC_TOP, JSPS_AC_RIGHT и JSPS_AC_BOTTOM. В остальных случаях он будет проигнорирован.

Функция jspsAlignContainer не возвращает значения.

Листинг 10.1. Функция jspsAlignContainer, выполняющая выравнивание свободно позиционируемого элемента вдоль заданного края родителя

```
var JSPS_AC_LEFT = 0;
var JSPS_AC_LEFTTOP = 1;
var JSPS_AC_TOP = 2;
var JSPS_AC_RIGHTTOP = 3;
var JSPS_AC_RIGHT = 4;
var JSPS_AC_RIGHTBOTTOM = 5;
var JSPS_AC_BOTTOM = 6;
var JSPS_AC_LEFTBOTTOM = 7;
var JSPS_AC_CENTER = 8;
var JSPS_AC_FULL = 9;
```

```
function jspsAlignContainer(pContainer, pEdge, pDoResize)
  {
    if (typeof(pEdge) == "undefined") pEdge = JSPS_AC_CENTER;
    if (!pDoResize) pDoResize = false;
    var pParentElement = pContainer.parentNode;
    var parentWidth = pParentElement.clientWidth;
    var parentHeight = pParentElement.clientHeight;
    var containerWidth = pContainer.offsetWidth;
    var containerHeight = pContainer.offsetHeight;
    switch (pEdge)
      {
        case JSPS_AC_LEFT:
          pContainer.style.left = 0;
          if (pDoResize)
            {
              pContainer.style.top = 0;
              pContainer.style.height = parentHeight;
            }
          break;
        case JSPS_AC_LEFTTOP:
          pContainer.style.left = 0;
          pContainer.style.top = 0;
          break;
        case JSPS_AC_TOP:
          pContainer.style.top = 0;
          if (pDoResize)
            {
              pContainer.style.left = 0;
              pContainer.style.width = parentWidth;
            }
          break;
        case JSPS_AC_RIGHTTOP:
          pContainer.style.left = parentWidth - containerWidth;
          pContainer.style.top = 0;
          break;
        case JSPS_AC_RIGHT:
          pContainer.style.left = parentWidth - containerWidth;
          if (pDoResize)
            {
              pContainer.style.top = 0;
              pContainer.style.height = parentHeight;
            }
          break;
```

```
      case JSPS_AC_RIGHTBOTTOM:
        pContainer.style.left = parentWidth - containerWidth;
        pContainer.style.top = parentHeight - containerHeight;
        break;
      case JSPS_AC_BOTTOM:
        pContainer.style.top = parentHeight - containerHeight;
        if (pDoResize)
          {
            pContainer.style.left = 0;
            pContainer.style.width = parentWidth;
          }
        break;
      case JSPS_AC_LEFTBOTTOM:
        pContainer.style.left = 0;
        pContainer.style.top = parentHeight - containerHeight;
        break;
      case JSPS_AC_CENTER:
        pContainer.style.left = (parentWidth - containerWidth) / 2;
        pContainer.style.top = (parentHeight - containerHeight) / 2;
        break;
      case JSPS_AC_FULL:
        pContainer.style.left = 0;
        pContainer.style.top = 0;
        pContainer.style.width = parentWidth;
        pContainer.style.height = parentHeight;
        break;
    }
}
```

Хорошая идея!

Поместите объявление этой функции в файл сценариев aligncontainer.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="aligncontainer.js"></SCRIPT>
```

Пример

```
var contObject = document.all["cont"];
jspsAlignContainer(contObject, JSPS_AC_TOP, true);
```

Этот сценарий растянет элемент `cont` вдоль верхнего края его родителя.

Как выровнять один свободный элемент по краю другого?

Проблема

А если мне нужно выровнять свободно позиционируемый элемент не по краю его родителя, а по краю другого такого же элемента?

Решение

Тогда нужно использовать универсальную функцию `jspsAlignContainerByContainer` (листинг 10.2). Вот формат ее вызова:

```
jspsAlignContainerByContainer(<Выравниваемый элемент>,
<Край выравниваемого элемента>,
<Элемент, по которому выполняется выравнивание>,
<Край элемента, по которому выполняется выравнивание>
[, <Растягивать выравниваемый элемент>]);
```

Ну, с первым и третьим параметрами все должно быть ясно. Первый параметр задает выравниваемый элемент, а второй — элемент, по которому будет выполнено выравнивание. Естественно, оба элемента должны иметь одного родителя.

Второй параметр задает край выравниваемого элемента, которым он будет прижиматься к элементу, по которому выполняется выравнивание. Вот все доступные значения этого параметра (приведены имена псевдоконстант, значения которых можно найти в листинге 10.2):

- `JSPS_ACC_LEFT` — левый край;
- `JSPS_ACC_LEFTTOP` — левый верхний угол;
- `JSPS_ACC_TOP` — верхний край;
- `JSPS_ACC_RIGHTTOP` — правый верхний угол;
- `JSPS_ACC_RIGHT` — правый край;
- `JSPS_ACC_RIGHTBOTTOM` — правый нижний угол;
- `JSPS_ACC_BOTTOM` — нижний край;
- `JSPS_ACC_LEFTBOTTOM` — левый нижний угол.

Четвертый параметр задает край выравнивающего элемента, к которому будет прижиматься выравниваемый элемент. Здесь доступны те же самые значения, что приведены ранее для второго параметра.

Пятый параметр позволяет указать, растягивать ли выравниваемый элемент вдоль края элемента, по которому выполняется выравнивание. Если указано значение `true`, элемент будет растянут; значение `false` предписывает оста-

вить размеры элемента неизменными. Пропуск третьего параметра эквивалентен указанию значения `false`.

Нужно иметь в виду, что пятый параметр функции `jspsAlignContainerByContainer` имеет смысл только при значениях второго и четвертого параметров, равных `JSPS_AC_LEFT`, `JSPS_AC_TOP`, `JSPS_AC_RIGHT` и `JSPS_AC_BOTTOM`. В остальных случаях он будет проигнорирован.

Функция `jspsAlignContainerByContainer` не возвращает значения.

Листинг 10.2. Функция `jspsAlignContainerByContainer`, выполняющая выравнивание одного свободно позиционируемого элемента вдоль края другого

```
var JSPS_ACC_LEFT = 0;
var JSPS_ACC_LEFTTOP = 1;
var JSPS_ACC_TOP = 2;
var JSPS_ACC_RIGHTTOP = 3;
var JSPS_ACC_RIGHT = 4;
var JSPS_ACC_RIGHTBOTTOM = 5;
var JSPS_ACC_BOTTOM = 6;
var JSPS_ACC_LEFTBOTTOM = 7;

function jspsAlignContainerByContainer(pContainer, pEdge, pAligner,
↳pAlignerEdge, pDoResize)
  {
    if (!pDoResize) pDoResize = false;
    if (pContainer.parentNode == pAligner.parentNode)
      {
        var leftCoord = pContainer.offsetLeft;
        var topCoord = pContainer.offsetTop;
        switch (pAlignerEdge)
          {
            case JSPS_ACC_LEFT:
              leftCoord = pAligner.offsetLeft;
              break;
            case JSPS_ACC_LEFTTOP:
              leftCoord = pAligner.offsetLeft;
              topCoord = pAligner.offsetTop;
              break;
            case JSPS_ACC_TOP:
              topCoord = pAligner.offsetTop;
              break;
            case JSPS_ACC_RIGHTTOP:
              leftCoord = pAligner.offsetLeft + pAligner.offsetWidth;
```

```
        topCoord = pAligner.offsetTop;
        break;
      case JSPS_ACC_RIGHT:
        leftCoord = pAligner.offsetLeft + pAligner.offsetWidth;
        break;
      case JSPS_ACC_RIGHTBOTTOM:
        leftCoord = pAligner.offsetLeft + pAligner.offsetWidth;
        topCoord = pAligner.offsetTop + pAligner.offsetHeight;
        break;
      case JSPS_ACC_BOTTOM:
        topCoord = pAligner.offsetTop + pAligner.offsetHeight;
        break;
      case JSPS_ACC_LEFTBOTTOM:
        leftCoord = pAligner.offsetLeft;
        topCoord = pAligner.offsetTop + pAligner.offsetHeight;
        break;
    }
  switch (pEdge)
    {
      case JSPS_ACC_LEFT:
        if (pDoResize)
          {
            topCoord = pAligner.offsetTop;
            pContainer.style.height = pAligner.offsetHeight;
          }
        break;
      case JSPS_ACC_TOP:
        if (pDoResize)
          {
            leftCoord = pAligner.offsetLeft;
            pContainer.style.width = pAligner.offsetWidth;
          }
        break;
      case JSPS_ACC_RIGHTTOP:
        if (leftCoord != pContainer.offsetLeft)
          leftCoord -= pContainer.offsetWidth;
        break;
      case JSPS_ACC_RIGHT:
        if (leftCoord != pContainer.offsetLeft)
          leftCoord -= pContainer.offsetWidth;
        if (pDoResize)
          {
            topCoord = pAligner.offsetTop;
```

```
            pContainer.style.height = pAligner.offsetHeight;
          }
        break;
      case JSPS_ACC_RIGHTBOTTOM:
        if (leftCoord != pContainer.offsetLeft)
          leftCoord -= pContainer.offsetWidth;
        if (topCoord != pContainer.offsetTop)
          topCoord -= pContainer.offsetHeight;
        break;
      case JSPS_ACC_BOTTOM:
        if (leftCoord != pContainer.offsetLeft)
          leftCoord -= pContainer.offsetWidth;
        if (topCoord != pContainer.offsetTop)
          topCoord -= pContainer.offsetHeight;
        if (pDoResize)
          {
            leftCoord = pAligner.offsetLeft;
            pContainer.style.width = pAligner.offsetWidth;
          }
        break;
      case JSPS_ACC_LEFTBOTTOM:
        if (topCoord != pContainer.offsetTop)
          topCoord -= pContainer.offsetHeight;
        break;
    }
  pContainer.style.left = leftCoord;
  pContainer.style.top = topCoord;
 }
}
```

Хорошая идея!

Добавьте объявление этой функции в файл сценариев aligncontainer.js, содержащий объявление функции `jspsAlignContainer`. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="aligncontainer.js"></SCRIPT>
```

Пример

```
var cont1Object = document.all["cont1"];
var cont2Object = document.all["cont2"];
jspsAlignContainerByContainer(cont1Object, JSPS_ACC_TOP,
↳cont2Object, LSPS_ACC_BOTTOM, false);
```

Этот сценарий выровняет верхний край элемента `cont1` вдоль нижнего края элемента `cont2`.

Как сделать так, чтобы свободно позиционируемые элементы меняли свое расположение при изменении размеров окна Web-обозревателя?

Проблема

Я создал довольно сложный Web-дизайн на основе свободно позиционируемых элементов, и теперь мне нужно сделать так, чтобы они меняли свои размеры и местоположение, подстраиваясь под размеры окна Web-обозревателя. Как это выполнить?

Решение

Нет ничего проще! Далее приведена последовательность действий для такого случая.

1. Создаем набор свободно позиционируемых элементов и наполняем их содержимым.
2. Объявляем функцию-обработчик события `onResize` окна Web-обозревателя. Эта функция будет заниматься позиционированием элементов при изменении размеров окна. Привязываем эту функцию к событию `onResize`.
3. В самом конце кода Web-страницы вставляем вызов объявленной ранее функции-обработчика. Это нужно для того, чтобы все созданные нами свободно позиционируемые элементы правильно размещались на странице сразу же после ее открытия.

В теле функции-обработчика события `onResize` окна Web-обозревателя можно использовать описанные ранее функции `jspsAlignContainer` (см. листинг 10.1) и `jspsAlignContainerByContainer` (см. листинг 10.2).

> ***Народ предупреждает!***
>
> Выравнивание по краям окна Web-обозревателя в подобных функциях-обработчиках следует использовать осторожно. Дело в том, что при изменении размеров окна вдоль его краев на мгновение появляются полосы прокрутки, которые потом пропадают. Так вот, судя по всему, размеры окна, возвращаемые свойствами `document.body.clientWidth` и `document.body.clientHeight`, вычисляются с учетом этих полос прокрутки, которые к данному моменту уже отсутствуют. Все это приводит к тому, что свободные элементы располагаются не вплотную к краю окна Web-обозревателя, а на некотором расстоянии от него. Такой эффект чаще всего наблюдается в Firefox, иногда — в Opera; Internet Explorer же от этого недостатка избавлен.

В качестве решения этой проблемы можно порекомендовать вставить в стиль переопределения тега <BODY> вот такой атрибут:

```
overflow: scroll;
```

После этого в окне Web-обозревателя всегда будут присутствовать полосы прокрутки, и размеры окна всегда будут вычисляться корректно. Но такое окно выглядит не очень презентабельно — создается впечатление, будто страницу не доделали.

Так что для позиционирования элементов страницы в окне Web-обозревателя все-таки лучше использовать старые проверенные средства: фреймы, таблицы разметки и *"плавающие" контейнеры* (обычные контейнеры <DIV>, прижимающиеся к заданному краю страницы; такой эффект достигается с помощью особых атрибутов стилей). Уж с ними-то таких "чудес" не происходит. А свободно позиционируемые элементы оставить для создания специальных эффектов (графический курсор мыши, анимация, всплывающие предупреждения и пр.).

Пример

Далее приведен HTML-код страницы, содержащей свободно позиционируемый элемент. Этот элемент будет располагаться в центре страницы при любом изменении размеров окна Web-обозревателя. Для его выравнивания по центру страницы используется функция `jspsAlignContainer` (см. листинг 10.1).

```
<HTML>
  <HEAD>
    <TITLE>Всегда в центре страницы</TITLE>
    <SCRIPT SRC="aligncontainer.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsAlignContainer
    помещено в файл сценария aligncontainer.js -->
    <SCRIPT TYPE="text/javascript">
      function windowOnResize()
        {
          var contObject = document.all["cont"];
          jspsAlignContainer(contObject, JSPS_AC_CENTER);
        }
    </SCRIPT>
    <STYLE>
      #cont { background-color: #CCCCCC;
              position: absolute;
              left: 100px;
              top: 100px;
              width: 100px;
              height: 100px; }
    </STYLE>
  </HEAD>
```

```
  <BODY>
    <DIV ID="cont">Этот элемент всегда располагается в центре
    ↳страницы.</DIV>
    <SCRIPT TYPE="text/javascript">
      window.onresize = windowOnResize;
      windowOnResize();
    </SCRIPT>
  </BODY>
</HTML>
```

Создание анимации с помощью свободно позиционируемых элементов

Одна из замечательных возможностей, предлагаемых свободно позиционируемыми элементами, — это возможность задавать их размеры и местоположение. (Собственно, что об этом говорить — мы уже убедились, что все это правда.) Значит, мы можем создать на своих страницах настоящую анимацию. Этим-то мы как раз сейчас и займемся.

> ***Народ предупреждает!***
>
> Анимировать (т. е. заставить двигаться по странице) можно только свободно позиционируемый элемент.

Как создать простейшую анимацию?

Проблема

Мне нужно создать простую анимацию на Web-странице — свободный элемент, движущийся по прямой. Как это сделать?

Решение

Основной принцип создания анимации несложен. Давайте его рассмотрим.

Первым делом нам нужно создать свободно позиционируемый элемент, который мы будем анимировать. Он может содержать изображение, фрагмент текста, заголовок или что-то еще. Располагаться этот свободный элемент должен в начальной позиции, из которой он станет двигаться. Позиционировать его там можно как с помощью соответствующих атрибутов стилей, так и специально написанным сценарием, который должен находиться сразу же после HTML-кода, создающего этот элемент.

Далее нам нужно написать функцию, реализующую анимацию. Эта функция должна вычислять новые координаты свободного элемента и позиционировать его по этим координатам. Выполнив очередной "шаг" анимированного свободного элемента, она должна завершать свою работу.

Напоследок нам останется написать сценарий, который запустит анимацию. Он может запускаться после загрузки Web-страницы (обработчик события `onLoad` тела Web-страницы), в ответ на щелчок по кнопке или на гиперссылке или еще как-то — в данный момент это неважно. А важно то, что этот сценарий должен содержать вызов метода `setInterval` объекта `window`, создающего таймер. В качестве параметров этого метода мы передадим объявленную ранее функцию, которая будет реализовывать анимацию, и интервал времени, через который она будет вызываться.

Вот и все. Теперь написанная нами функция будет периодически вызываться, вычислять новые координаты свободного элемента и перемещать его по этим координатам. Результат — элемент будет двигаться по определенной *траектории* (так называется путь, по которому он должен двигаться).

> ***Народ замечает***
>
> Вообще, свободный элемент совсем не обязательно должен именно двигаться. Он может изменять свои размеры, менять цвет или пропадать со страницы и снова появляться. Нечто подобное автор книги уже описал в *главе 5*.

Если мы хотим, чтобы наша анимация воспроизводилась бесконечно (в смысле, пока открыта данная страница — такая анимация называется *зацикленной*), нам больше ничего делать не нужно. Если же мы хотим, чтобы анимация воспроизводилась до какого-то определенного момента, то должны будем совершить дополнительные действия. Как мы помним из *главы 5*, для удаления таймера нужно вызвать метод `clearInterval` объекта `window`, передав ему идентификатор данного таймера. Этого будет достаточно, чтобы наша анимация завершилась.

Осталось только решить, где вызвать метод `clearInterval` объекта `window`. Это можно сделать в функции, создающей анимацию, организовав в ее теле проверку на достижение свободным элементом конечной точки его траектории. Также можно поместить вызов этого метода в функцию-обработчик события `onClick` кнопки или гиперссылки. Возможны и другие варианты.

> ***Народ предупреждает!***
>
> Анимацию на Web-страницах следует использовать очень и очень осторожно. Движущиеся по странице элементы сейчас считаются признаком непрофессионализма Web-дизайнера и вообще плохим тонов в Web-дизайне. И это уже не говоря о том, что анимация будет отвлекать и раздражать посетителя сайта.

Пример 1

Далее приведен HTML-код Web-страницы, содержащей анимированный элемент — графическое изображение. Это изображение будет двигаться из левого нижнего в правый верхний угол страницы и обратно.

```
<HTML>
  <HEAD>
    <TITLE>Анимированный элемент</TITLE>
    <!-- Определяем стиль для анимированного элемента. Задавать реальные
    начальные координаты необязательно — все равно мы позиционируем наш
    элемент программно -->
    <STYLE>
      #img { position: absolute;
             left: 100px;
             top: 100px; }
    </STYLE>
  </HEAD>
  <BODY>
    <!-- Анимированный элемент -->
    <IMG ID="img" SRC="image.gif">
    <SCRIPT TYPE="text/javascript">
      var imgObject = document.images["img"];
      // Временной интервал для таймера
      var int = 100;
      // Приращение изменения горизонтальной координаты. Меняя его
      // и временной интервал, мы можем управлять плавностью
      // воспроизведения анимации
      var d = 10;
      // Начальные и конечные координаты анимированного элемента
      var x1 = 0;
      var y1 = document.body.clientHeight - imgObject.offsetHeight - 1;
      var x2 = document.body.clientWidth - imgObject.offsetWidth - 1;
      var y2 = 0;
      // Граничные значения горизонтальной координаты. Мы используем их
      // в проверке достижения начала и конца траектории анимированного
      // элемента
      if (x1 > x2)
        {
          var xmin = x2;
          var xmax = x1;
        }
      else
        {
          var xmin = x1;
          var xmax = x2;
        }
      // Корректируем граничные координаты на величину приращения
      xmin += d;
      xmax -= d;
```

```
      var k = Math.abs((y2 - y1) / (x2 - x1));
      if (y1 > y2) k = -k;
      // Текущая горизонтальная координата анимированного элемента
      var x = x1;

      // Функция, реализующая анимацию
      function doStep()
        {
          // Изменяем текущую координату на величину приращения
          if (x2 > x1)
            x += d
          else
            x -= d;
          // Перемещаем анимированный элемент на новое место
          imgObject.style.left = x;
          imgObject.style.top = y1 + k * Math.abs(x - x1);
          // Проверяем, не дошел ли анимированный элемент до начала или
          // конца траектории. Если это так, меняем знак величины
          // приращения
          if ((x < xmin) || (x > xmax))
            d = -d;
        }

      // Устанавливаем анимированный элемент в начальную точку траектории
      imgObject.style.left = x1;
      imgObject.style.top = y1;
      // Создаем таймер. Воспроизведение анимации началось!
      window.setInterval(doStep, int);
    </SCRIPT>
  </BODY>
</HTML>
```

Кстати, функция `doStep`, реализующая анимацию, в этом примере является, можно сказать, универсальной. Мы можем задать для начальных и конечных координат анимированного элемента любые значения, и она будет работать.

Единственное исключение — она не работает, если анимированный элемент должен двигаться по вертикали. В самом деле, давайте рассмотрим выражение

```
var k = Math.abs((y2 - y1) / (x2 - x1));
```

Если траектория анимированного элемента строго вертикальна, то горизонтальные координаты ее начальной и конечной точек (`x1` и `x2`) одинаковы. Тогда в этом выражении возникнет ошибка — деление на ноль, Web-обозреватель выдаст сообщение об ошибке и выполнение сценария прервется.

Пример Web-страницы с анимированным элементом, движущимся по вертикальной траектории, приведен далее.

Пример 2

Приведенный в предыдущем разделе пример Web-страницы с анимированным элементом не будет работать, если для этого элемента задать вертикальную траекторию (т. е. равные значения переменных `x1` и `x2`). Давайте рассмотрим другой, работающий, пример страницы, анимированный элемент которой движется по вертикали.

```
<HTML>
  <HEAD>
    <TITLE>Анимированный элемент</TITLE>
    <!-- Определяем стиль для анимированного элемента. Задавать реальные
    начальные координаты необязательно — все равно мы позиционируем наш
    элемент программно -->
    <STYLE>
      #img { position: absolute;
             left: 100px;
             top: 100px; }
    </STYLE>
  </HEAD>
  <BODY>
    <!-- Анимированный элемент -->
    <IMG ID="img" SRC="image.gif">
    <SCRIPT TYPE="text/javascript">
      var imgObject = document.images["img"];
      // Временной интервал для таймера
      var int = 100;
      // Приращение изменения вертикальной координаты. Меняя его и
      // временной интервал, мы можем управлять плавностью
      // воспроизведения анимации
      var d = 10;
      // Начальные и конечные координаты анимированного элемента.
      // Заметим, что горизонтальная координата одна и та же для обеих
      // точек траектории (в нашем случае это центр страницы)
      var x = (document.body.clientWidth - imgObject.offsetWidth - 1) /
      ↳2;
      var y1 = document.body.clientHeight - imgObject.offsetHeight - 1;
      var y2 = 0;
      // Граничные значения вертикальной координаты. Мы используем их
      // в проверке достижения начала и конца траектории анимированного
      // элемента
```

```
      if (y1 > y2)
        {
          var ymin = y2;
          var ymax = y1;
        }
      else
        {
          var ymin = y1;
          var ymax = y2;
        }
      // Корректируем граничные координаты на величину приращения
      ymin += d;
      ymax -= d;
      // Текущая вертикальная координата анимированного элемента
      var y = y1;

      // Функция, реализующая анимацию
      function doStep()
        {
          // Изменяем текущую координату на величину приращения
          if (y2 > y1)
            y += d
          else
            y -= d;
          // Перемещаем анимированный элемент на новое место. Здесь мы
          // задаем только вертикальную координату, так как
          // горизонтальная не меняется
          imgObject.style.top = y;
          // Проверяем, не дошел ли анимированный элемент до начала или
          // конца траектории. Если это так, меняем знак величины
          // приращения
          if ((y < ymin) || (y > ymax))
            d = -d;
        }

      // Устанавливаем анимированный элемент в начальную точку траектории
      imgObject.style.left = x;
      imgObject.style.top = y1;
      // Создаем таймер. Воспроизведение анимации началось!
      window.setInterval(doStep, int);
    </SCRIPT>
  </BODY>
</HTML>
```

Пример 3

А вот пример Web-страницы, анимированный элемент которой движется по окружности.

```
<HTML>
  <HEAD>
    <TITLE>Анимированный элемент</TITLE>
    <!-- Определяем стиль для анимированного элемента. Задавать реальные
    начальные координаты необязательно — все равно мы позиционируем наш
    элемент программно -->
    <STYLE>
      #img { position: absolute;
             left: 100px;
             top: 100px; }
    </STYLE>
  </HEAD>
  <BODY>
    <!-- Анимированный элемент -->
    <IMG ID="img" SRC="image.gif">
    <SCRIPT TYPE="text/javascript">
      var imgObject = document.images["img"];
      // Временной интервал для таймера
      var int = 100;
      // Приращение изменения угла поворота. Меняя его и временной
      // интервал, мы можем управлять плавностью воспроизведения анимации
      var d = .2;
      // Радиус окружности, по которой движется анимированный элемент
      var R = 100;
      // Координаты центра этой окружности (в нашем случае — центр
      // страницы)
      var x = (document.body.clientWidth - imgObject.offsetWidth - 1) /
      ↳2;
      var y = (document.body.clientHeight - imgObject.offsetHeight - 1) /
      ↳2;
      // Граничное условие
      var rr = Math.PI * 2;
      // Текущий угол поворота
      var a = 0;

      // Функция, реализующая анимацию
      function doStep()
        {
          // Изменяем текущий угол поворота на величину приращения
          a += d;
```

```
        // Перемещаем анимированный элемент на новое место
        imgObject.style.left = x + R * Math.sin(a);
        imgObject.style.top = y + R * Math.cos(a);
        // Проверяем, не совершил ли анимированный элемент полный
        // поворот. Если это так, обнуляем величину текущего угла
        // поворота
        if ((a > rr - d) && (a < rr + d))
          a = 0;
      }

    // Устанавливаем анимированный элемент в начальную точку траектории
    imgObject.style.left = x;
    imgObject.style.top = y + R / 2;
    // Создаем таймер. Воспроизведение анимации началось!
    window.setInterval(doStep, int);
  </SCRIPT>
 </BODY>
</HTML>
```

Как создать более сложную анимацию?

Проблема

Мне нужно создать анимированный элемент, который двигался бы по ломаной линии. Как это сделать?

Решение

Проще всего использовать универсальный объект `ComplexAnimation` (листинг 10.3), который предназначен именно для создания подобной анимации. Вот формат вызова его конструктора:

```
ComplexAnimation(<Анимированный элемент>, <Массив ключевых точек>
[, <Зациклить анимацию>]);
```

Первым параметром в конструктор передается элемент страницы, который нужно анимировать. Этот элемент необязательно должен быть свободно позиционируемым — сам объект `ComplexAnimation` позаботится об этом.

А вот второй параметр конструктора нужно рассмотреть подробнее. Давайте этим и займемся.

Прежде всего, нужно сказать, что для задания траектории движения анимированного элемента используются так называемые *ключевые точки*. Эти точки задают начало и конец каждого отрезка траектории (вспомним, что она представляет собой ломаную линию), а также момент времени, в который анимированный элемент должен находиться в каждой из этих точек. Ключе-

вых точек должно быть столько, чтобы описать всю траекторию движения анимированного элемента — не больше и не меньше.

Каждый элемент *Массива ключевых точек* представляет собой другой, вложенный, массив, содержащий три элемента. Эти элементы задают:

- горизонтальную координату точки траектории;
- вертикальную координату точки траектории;
- момент времени, в который анимированный элемент должен находиться в этой точке, заданный в секундах, прошедших с начала анимации.

Первый элемент *Массива ключевых точек* задает начало траектории. Первые два элемента задают координаты начальной точки траектории, а третий — момент времени — должен быть равным нулю (хотя это и не обязательно). Второй элемент *Массива ключевых точек* задает конец первого отрезка траектории и одновременно начало второго и т. д. Последний элемент *Массива ключевых точек* задает последнюю точку траектории; в ней движение анимированного элемента закончится.

Массив ключевых точек может содержать сколько угодно элементов, но должны присутствовать хотя бы два.

Нужно иметь в виду, что *Массив ключевых точек* должен существовать все время, пока существует страница (т. е. быть глобальным). Дело в том, что объект `ComplexAnimation` использует его для создания анимации и, если этот массив оказался недоступным (например, если мы его удалим), не сможет правильно ее отобразить.

Третий, необязательный, параметр конструктора объекта `ComplexAnimation` позволяет зациклить анимацию. Зацикливает ее значение `true` этого параметра; значение же `false` делает анимацию "конечной" (анимированный элемент, дойдя до конца траектории, остановится). Пропуск третьего параметра эквивалентен указанию значения `false`.

Для задания интервала таймера объект `ComplexAnimation` использует переменную `jsps_CA_interval`. Эта переменная объявляется в листинге 10.3 перед объявлением данного объекта и содержит значения 10 (т. е. "шаги" анимации будут происходить каждые 10 миллисекунд). Если кому-то из Web-программистов, использующих этот объект, потребуется другой интервал, он просто присвоит нужное значение данной переменной.

При вызове конструктора объекта `ComplexAnimation` создается только экземпляр этого объекта, но анимация на воспроизведение не запускается. Для ее запуска нужно вызвать не принимающий параметров и не возвращающий значения метод `start` этого объекта. А метод `stop` позволяет остановить воспроизведение анимации; он также не принимает параметров и не возвращает результата.

Метод `reset` объекта `ComplexAnimation` перезапускает анимацию, т. е. останавливает ее и помещает анимированный элемент в начальную точку траектории. После вызова этого метода следует запустить анимацию, вызвав метод `start`. Метод `reset` также не принимает параметров и не возвращает результата.

Свойство `element` объекта `ComplexAnimation` возвращает анимированный элемент. Оно может пригодиться, например, для изменения внешнего вида или содержимого этого элемента во время его движения.

Свойство `phaseNum` этого же объекта возвращает номер последней пройденной ключевой точки. Нумерация ключевых точек начинается с нуля; если же ни одна ключевая точка не была пройдена (например, если мы создали экземпляр объекта `ComplexAnimation`, но еще не запустили анимацию), оно вернет значение –1.

Свойство `onPhasePass` позволяет создать обработчик одноименного события объекта `ComplexAnimation`. Это событие возникает при проходе очередной ключевой точки траектории. Соответствующую функцию-обработчик нужно просто присвоить этому свойству (собственно, мы об этом уже знаем). Эта функция должна принимать единственный параметр — экземпляр объекта `ComplexAnimation`, в котором произошло событие.

Обработчик события `onPhasePass` можно использовать для различных целей. Например, мы можем при прохождении очередной ключевой точки запускать или останавливать другую анимацию. Также мы можем менять внешний вид или содержимое анимированного элемента.

Народ предупреждает!

Перед уничтожением экземпляра объекта `ComplexAnimation` следует вызвать метод `stop` этого объекта.

Листинг 10.3. Объект `ComplexAnimation`, позволяющий создать анимацию

```
// Объявляем переменную jsps_CA_interval, хранящую временной интервал
// для таймера
var jsps_CA_interval = 10;

// Счетчик созданных к данному времени экземпляров объекта
// ComplexAnimation
var jsps_CA_counter = 0;
// Список всех созданных к данному времени экземпляров объекта
// ComplexAnimation
var jsps_CA_list = new Array();
```

```
// Функция-конструктор
function ComplexAnimation(pElement, pPhases, pIsLooped)
  {
    // Объявляем общедоступные свойства и методы
    if (!pIsLooped) pIsLooped = false;
    this.element = pElement;
    this.phases = pPhases;
    this.isLooped = pIsLooped;
    this.start = mjspsCAStart;
    this.stop = mjspsCAStop;
    this.reset = mjspsCAReset;
    this.onPhasePass = null;
    // Объявляем различные служебные свойства
    this.timer = null;
    // Номер данного экземпляра объекта в списке jsps_CA_list
    this.myNumber = jsps_CA_counter;
    // Заносим данный экземпляр объекта в список jsps_CA_list
    jsps_CA_list[jsps_CA_counter] = this;
    // Увеличиваем счетчик экземпляров этого объекта на единицу,
    // подготавливая его для создания нового экземпляра
    jsps_CA_counter++;
    this.reset();
  }

// Функция, реализующая метод start
function mjspsCAStart()
  {
    // Перед созданием таймера проверяем, не закончилась ли анимация
    if (this.phaseNum < this.phases.length - 1)
      this.timer = window.setInterval("jspsCADoAction(" +
      ↳this.myNumber.toString() + ")", jsps_CA_interval);
  }

// Функция, реализующая метод stop
function mjspsCAStop()
  {
    if (this.timer)
      window.clearInterval(this.timer);
    this.timer = null;
  }

// Функция, реализующая метод reset
function mjspsCAReset()
```

```
  {
    // Останавливаем анимацию
    this.stop();
    // Делаем анимируемый элемент страницы свободно позиционируемым
    if (this.element.style.position != "absolute")
      this.element.style.position = "absolute";
    // Устанавливаем его в начальную точку траектории
    this.element.style.left = this.phases[0][0];
    this.element.style.top = this.phases[0][1];
    // Объявляем служебные свойства
    this.phaseNum = -1;
    this.time = 0;
  }

// Функция, реализующая анимацию
function jspsCADoAction(myNumber)
  {
    var my = jsps_CA_list[myNumber];
    // Запускаем анимацию, если она еще не запущена
    if (my.phaseNum == -1)
      {
        my.phaseNum = 0;
        if (my.onPhasePass) my.onPhasePass(my);
      }
    // Если анимация не закончилась, реализуем очередной "шаг"
    if (my.phaseNum < my.phases.length - 1)
      {
        // Получаем координаты начальной и конечной точки очередного
        // отрезка траектории и их временные отметки
        var x1 = my.phases[my.phaseNum][0];
        var y1 = my.phases[my.phaseNum][1];
        var x2 = my.phases[my.phaseNum + 1][0];
        var y2 = my.phases[my.phaseNum + 1][1];
        var time1 = (my.phaseNum == 0) ? 0 :
        ↳my.phases[my.phaseNum][2] * 1000;
        var time2 = my.phases[my.phaseNum + 1][2] * 1000;
        // Проверяем, не дошел ли анимированный элемент до конечной точки
        // данного отрезка траектории
        if ((my.time > time2 - jsps_CA_interval) && (my.time < time2 +
        ↳jsps_CA_interval))
          {
            // Если дошел, начинаем движение по следующему отрезку
            my.time = time2;
```

```
          my.phaseNum++;
          if (my.onPhasePass) my.onPhasePass(my);
        }
      else
        {
          // Иначе реализуем очередной "шаг" по текущему отрезку
          if (x1 == x2)
            {
              var yd = Math.abs((my.time - time1) * (y2 - y1) /
              ↳(time2 - time1));
              if (y1 > y2) yd = -yd;
              my.element.style.top = y1 + yd;
            }
          else
            {
              var xd = Math.abs((my.time - time1) * (x2 - x1) /
              ↳(time2 - time1));
              if (x1 > x2) xd = -xd;
              var k = Math.abs((y2 - y1) / (x2 - x1));
              if (y1 > y2) k = -k;
              var x = x1 + xd;
              my.element.style.left = x;
              my.element.style.top = y1 + k * Math.abs(x - x1);
            }
          my.time += jsps_CA_interval;
        }
    }
  else
    // Если анимация закончилась и если анимация зациклена,
    // запускаем ее с начала
    if (my.isLooped)
      {
        my.reset();
        my.start();
      }
}
```

Хорошая идея!

Поместите объявление этого объекта в файл сценариев complexanimation.js. Впоследствии, чтобы использовать его, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="complexanimation.js"></SCRIPT>
```

Пример

Далее приведен HTML-код Web-страницы, содержащей анимированный элемент, который движется по прямоугольной траектории.

```
<HTML>
  <HEAD>
    <TITLE>Анимированный элемент</TITLE>
    <SCRIPT SRC="complexanimation.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта ComplexAnimation было
    помещено в файл сценариев complexanimation.js -->
  </HEAD>
  <BODY>
    <IMG ID="img" SRC="image.gif">
    <SCRIPT TYPE="text/javascript">
      var phasesArray = new Array();
      phasesArray[0] = new Array(100, 100, 0);
      phasesArray[1] = new Array(100, 200, 3);
      phasesArray[2] = new Array(200, 200, 6);
      phasesArray[3] = new Array(200, 100, 9);
      phasesArray[4] = new Array(100, 100, 12);
      var imgObject = document.all["img"];
      var caObject = new ComplexAnimation(imgObject, phasesArray, true);
      caObject.start();
    </SCRIPT>
  </BODY>
</HTML>
```

Как видим, код получился очень компактным, хотя анимация, которую мы реализовали, весьма сложна. Вот что значит использовать универсальные функции и объекты!

Заметим, как мы задали моменты времени в массиве ключевых точек. Задавать их следует именно так — как количество секунд, прошедших с момента начала воспроизведения анимации. Если задать их как-то иначе (например, как количество секунд, прошедших с начала движения по данному отрезку траектории), объект `ComplexAnimation` будет работать некорректно (если вообще будет), либо мы получим сообщение об ошибке от Web-обозревателя.

Народ предупреждает!

Народ предупреждает еще раз — анимацию на Web-страницах следует использовать с большой оглядкой. Как правило, она ничего не дает для посетителя, но отвлекает и раздражает его.

Народная мудрость

Лучший Web-дизайн — это тот, которого не существует.

Комментарий автора книги

Народные мудрости не всегда ясны, но всегда парадоксальны. В данном случае народ хотел сказать: лучший Web-дизайн — это тот, который незаметен с точки зрения посетителя сайта, который не мешает посетителю читать текст и рассматривать картинки на страницах. Скажем так — хороший Web-дизайн ненавязчив.

Как создать графический курсор мыши?

Проблема

Я видел на одном сайте *графический курсор мыши* (небольшое изображение, которое перемещается вслед за системным курсором мыши и, собственно, играет его роль). Как мне сделать такой же?

Решение

Ну, это не самый сложный пример анимации. Нам достаточно написать функцию-обработчик события `onMouseMove` тела Web-страницы, которая будет выяснять координаты курсора мыши и перемещать прямо под него нужное нам свободно позиционируемое изображение.

Было бы также неплохо привязать ту же самую функцию-обработчик к событию `onLoad` тела страницы. Это нужно, чтобы правильно позиционировать графический курсор сразу же после ее загрузки.

Пример

Далее приведен HTML-код Web-страницы, реализующей графический курсор мыши, изображение которого хранится в файле cursor.gif.

```
<HTML>
  <HEAD>
    <TITLE>Графический курсор мыши</TITLE>
    <SCRIPT SRC="browserdetect.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsGetProgramInfo было
    помещено в файл сценариев browserdetect.js -->
    <SCRIPT TYPE="text/javascript">
      function bodyOnMouseMoveIEOpera()
        {
          imgObject.style.left = window.event.clientX;
          imgObject.style.top = window.event.clientY;
        }

      function bodyOnMouseMoveFF(evt)
        {
          imgObject.style.left = evt.clientX;
```

```
        imgObject.style.top = evt.clientY;
      }
    </SCRIPT>
    <STYLE>
      #img { position: absolute; }
    </STYLE>
  </HEAD>
  <BODY>
    <P>Это Web-страница с графическим курсором мыши.</P>
    <!-- Создаем свободно позиционируемое изображение, которое станет
    графическим курсором мыши -->
    <IMG ID="img" SRC="cursor.gif">
    <SCRIPT TYPE="text/javascript">
      var imgObject = document.all["img"];
      var piObject = jspsGetProgramInfo();
      if ((piObject.programName == JSPS_GPI_MSIE) ||
      ↳(piObject.programName == JSPS_GPI_OPERA))
        {
          document.body.onmousemove = bodyOnMouseMoveIEOpera;
          document.body.onload = bodyOnMouseMoveIEOpera;
        }
      else
        {
          document.body.addEventListener("mousemove", bodyOnMouseMoveFF,
          ↳false);
          document.body.addEventListener("load", bodyOnMouseMoveFF,
          ↳false);
        }
    </SCRIPT>
  </BODY>
</HTML>
```

Внимание!

Приведенный пример использует функцию `jspsGetProgramInfo`, чье объявление приведено в листинге 3.1.

Здесь мы в который уже раз столкнулись с несовместимостью Web-обозревателей. Мы помним, что Internet Explorer и Opera сами передают в функции-обработчики экземпляр объекта `event`, содержащий всю информацию о событии; Firefox же делает это только в случае функций-слушателей. А ведь нам нужно в теле функции-обработчика получить координаты курсора мыши. Поэтому нам пришлось писать две функции-обработчика (для Internet Explorer и Opera и для Firefox) и привязывать их к событиям по-разному (как обычный обработчик события и как функцию-слушатель).

Народ предупреждает!

Судя по всему, в Opera и Firefox событие `onMouseMove` тела страницы нормально генерируется только в том случае, если курсор мыши перемещается над содержимым страницы, но не над пустым ее пространством. Это нужно иметь в виду.

Народ советует

Вообще, графический курсор мыши — абсолютно бесполезная вещь. Использовать ее на Web-страницах незачем. (Пожалуй, единственное исключение — приложения, созданные на основе Web-страниц; но даже в таком случае он пригодится не всегда.)

Как реализовать drag'n'drop?

Проблема

Мне очень нужно сделать некоторые элементы страницы перетаскиваемыми, т. е. реализовать технологию drag'n'drop. Как это сделать?

Решение

Ничего сложного! Далее приведена последовательность действий, которые нам следует выполнить.

1. Создаем элемент страницы, который должен быть перетаскиваемым. Разумеется, этот элемент должен быть свободно позиционируемым.
2. Объявляем глобальную переменную, которая будет хранить признак того, что элемент страницы в данный момент буксируется мышью. Присваиваем ей изначальное значение, говорящее о том, что признак буксировки отсутствует (т. е. элемент страницы стоит на месте). Проще всего для этого использовать логические значения.
3. Объявляем функцию-обработчик события `onMouseDown` элемента страницы, который должен быть перетаскиваемым. Эта функция будет заниматься тем, что устанавливать признак буксировки (присваивать соответствующее значение объявленной на шаге 2 переменной).
4. Объявляем функцию-обработчик события `onMouseMove` тела страницы. Эта функция будет, собственно, реализовывать перетаскивание элемента страницы. Сначала она проверит, установлен ли признак буксировки, и, если он установлен, задаст соответствующие координаты для перетаскиваемого элемента.
5. Объявляем функцию-обработчик события `onMouseUp` элемента страницы, который должен быть перетаскиваемым. Она сбросит признак буксировки.

После этого буксируемый элемент страницы перестанет быть буксируемым.

Об описанных выше событиях было подробно сказано в *главе 2*.

Мы можем позволить посетителю перетаскивать элементы страницы только в строго заданные места. Для этого в функции-обработчике события `onMouseMove` тела страницы мы должны проверить, находится ли курсор мыши в заданном месте страницы, и каким-то образом дать понять посетителю, что он может "бросить" здесь буксируемый элемент (например, изменить цвет его фона). Такую же проверку должна выполнять функция-обработчик события `onMouseUp` и, только если результат проверки положителен, сбрасывать признак буксировки.

Также можно при попытке перетащить буксируемый элемент на недопустимое место возвращать его в изначальное местоположение. Для этого нужно только сохранить его изначальные координаты в функции-обработчике события `onMouseDown`. Сам же возврат на изначальное место должен выполняться в функции — обработчике события `onMouseUp` после проверки, на допустимое ли место посетитель перетащил этот элемент.

Народ советует

Буксируемые элементы допустимы только в приложениях, созданных на основе Web-страниц. На обычных Web-страницах делать им совершенно нечего.

Пример 1

Далее приведен HTML-код Web-страницы, содержащей свободно позиционируемый элемент, который можно перетаскивать мышью.

```
<HTML>
  <HEAD>
    <TITLE>Drag'n'drop</TITLE>
    <SCRIPT SRC="browserdetect.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsGetProgramInfo было
    помещено в файл сценариев browserdetect.js -->
    <SCRIPT TYPE="text/javascript">
      // Объявляем переменную, хранящую признак буксировки
      var isDragging = false;
      // Объявляем переменные, хранящие координаты курсора мыши
      // относительно перетаскиваемого элемента страницы.
      // Они понадобятся, чтобы скорректировать положение
      // этого элемента относительно курсора мыши при буксировке
      var xd = 0;
      var yd = 0;
```

```
  // Функция-обработчик события onMouseDown перетаскиваемого элемента
  // для Internet Explorer и Opera
  function contOnMouseDownIEOpera()
    {
      isDragging = true;
      xd = window.event.offsetX;
      yd = window.event.offsetY;
    }

  // Функция-обработчик события onMouseDown перетаскиваемого
  // элемента для Firefox
  function contOnMouseDownFF(evt)
    {
      isDragging = true;
      xd = evt.clientX - contObject.offsetLeft;
      yd = evt.clientY - contObject.offsetTop;
    }

  // Функция-обработчик события onMouseMove тела страницы
  // для Internet Explorer и Opera
  function bodyOnMouseMoveIEOpera()
    {
      if (isDragging)
        {
          contObject.style.left = window.event.clientX - xd;
          contObject.style.top = window.event.clientY - yd;
        }
    }

  // Функция-обработчик события onMouseMove тела страницы
  // для Firefox
  function bodyOnMouseMoveFF(evt)
    {
      if (isDragging)
        {
          contObject.style.left = evt.clientX - xd;
          contObject.style.top = evt.clientY - yd;
        }
    }

  // Функция-обработчик события onMouseUp перетаскиваемого элемента
  function contOnMouseUp()
    {
      isDragging = false;
    }
</SCRIPT>
```

```
    <STYLE>
      #cont { position: absolute;
              left: 100px;
              top: 100px;
              width: 100px;
              height: 100px;
              background-color: #CCCCCC; }
    </STYLE>
  </HEAD>
  <BODY>
    <DIV ID="cont">Этот элемент страницы можно перетаскивать мышью.</DIV>
    <SCRIPT TYPE="text/javascript">
      var contObject = document.all["cont"];
      var piObject = jspsGetProgramInfo();
      if ((piObject.programName == JSPS_GPI_MSIE) ||
      ↳(piObject.programName == JSPS_GPI_OPERA))
        {
          contObject.onmousedown = contOnMouseDownIEOpera;
          document.body.onmousemove = bodyOnMouseMoveIEOpera;
        }
      else
        {
          contObject.addEventListener("mousedown", contOnMouseDownFF,
          ↳false);
          document.body.addEventListener("mousemove", bodyOnMouseMoveFF,
          ↳false);
        }
      contObject.onmouseup = contOnMouseUp;
    </SCRIPT>
  </BODY>
</HTML>
```

Внимание!

Приведенный пример использует функцию `jspsGetProgramInfo`, чье объявление приведено в листинге 3.1.

Да, и здесь проклятая несовместимость Web-обозревателей. И когда только их разработчики договорятся между собой!..

Пример 2

А вот пример Web-страницы со свободно позиционируемым элементом, который можно перетаскивать только в определенные места на странице. Такими местами являются два других свободно позиционируемых элемента.

```
<HTML>
  <HEAD>
    <TITLE>Drag'n'drop</TITLE>
    <SCRIPT SRC="browserdetect.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsGetProgramInfo было
    помещено в файл сценариев browserdetect.js -->
    <SCRIPT TYPE="text/javascript">
      // Объявляем переменную, хранящую признак буксировки
      var isDragging = false;
      // Объявляем переменные, хранящие координаты курсора мыши
      // относительно перетаскиваемого элемента страницы.
      // Они понадобятся, чтобы скорректировать
      // положение этого элемента относительно
      var xd = 0;
      var yd = 0;
      // Объявляем переменные для хранения изначальных координат
      // перетаскиваемого элемента
      var xbegin = 0;
      var ybegin = 0;

      // Функция, проверяющая, попал ли перетаскиваемый элемент
      // на один из элементов-"приемников"
      function isReceived(pReceiver)
        {
          return ((contObject.offsetLeft >= pReceiver.offsetLeft) &&
          ↳(contObject.offsetLeft + contObject.offsetWidth <=
          ↳pReceiver.offsetLeft + pReceiver.offsetWidth) &&
          ↳(contObject.offsetTop >= pReceiver.offsetTop) &&
          ↳(contObject.offsetTop + contObject.offsetHeight <=
          ↳pReceiver.offsetTop + pReceiver.offsetHeight));
        }

      // Функция-обработчик события onMouseDown перетаскиваемого
      // элемента для Internet Explorer и Opera
      function contOnMouseDownIEOpera()
        {
          isDragging = true;
          xd = window.event.offsetX;
          yd = window.event.offsetY;
          // Сохраняем изначальные координаты перетаскиваемого элемента
          xbegin = contObject.offsetLeft;
          ybegin = contObject.offsetTop;
        }
```

```
// Функция-обработчик события onMouseDown перетаскиваемого
// элемента для Firefox
function contOnMouseDownFF(evt)
  {
    isDragging = true;
    xd = evt.clientX - contObject.offsetLeft;
   yd = evt.clientY - contObject.offsetTop;
    // Сохраняем изначальные координаты перетаскиваемого элемента
    xbegin = contObject.offsetLeft;
    ybegin = contObject.offsetTop;
  }

// Функция-обработчик события onMouseMove тела страницы
// для Internet Explorer и Opera
function bodyOnMouseMoveIEOpera()
  {
    if (isDragging)
      {
        contObject.style.left = window.event.clientX - xd;
        contObject.style.top = window.event.clientY - yd;
        // Проверяем, попал ли перетаскиваемый элемент на один
        // из элементов-"приемников", и, если так, меняем цвет
        // его фона, сообщая тем самым посетителю,
        // что элемент можно "бросить"
        if ((isReceived(receiver1Object)) ||
        ↳(isReceived(receiver2Object)))
          contObject.style.backgroundColor = "#FF0000"
        else
          contObject.style.backgroundColor = "#CCCCCC";
      }
  }

// Функция-обработчик события onMouseMove тела страницы
// для Firefox
function bodyOnMouseMoveFF(evt)
  {
    if (isDragging)
      {
        contObject.style.left = evt.clientX - xd;
        contObject.style.top = evt.clientY - yd;
        // Проверяем, попал ли перетаскиваемый элемент на один
        // из элементов-"приемников", и, если так, меняем цвет
        // его фона, сообщая тем самым посетителю,
        // что элемент можно "бросить"
```

```
          if ((isReceived(receiver1Object)) ||
          ↳(isReceived(receiver2Object)))
            contObject.style.backgroundColor = "#FF0000"
          else
            contObject.style.backgroundColor = "#CCCCCC";
        }
    }

  // Функция-обработчик события onMouseUp перетаскиваемого элемента
  function contOnMouseUp()
    {
      isDragging = false;
      // Обязательно возвращаем перетаскиваемому элементу
      // прежний цвет фона
      contObject.style.backgroundColor = "#CCCCCC";
      // Если перетаскиваемый элемент был "брошен" на недопустимое
      // место, возвращаем его в прежнюю позицию
      if (!((isReceived(receiver1Object)) ||
      ↳(isReceived(receiver2Object))))
        {
          contObject.style.left = xbegin;
          contObject.style.top = ybegin;
        }
    }
</SCRIPT>
<STYLE>
  #receiver1 { position: absolute;
               left: 100px;
               top: 300px;
               width: 200px;
               height: 200px;
               background-color: #666666;
               color: #FFFFFF; }
  #receiver2 { position: absolute;
               left: 400px;
               top: 300px;
               width: 200px;
               height: 200px;
               background-color: #666666;
               color: #FFFFFF; }
  #cont      { position: absolute;
               left: 100px;
               top: 100px;
               width: 100px;
```

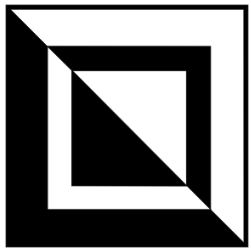
```
                height: 100px;
                background-color: #CCCCCC; }
    </STYLE>
  </HEAD>
  <BODY>
    <DIV ID="receiver1">Тащите его сюда!</DIV>
    <DIV ID="receiver2">Тащите его сюда!</DIV>
    <DIV ID="cont">Этот элемент страницы можно перетаскивать мышью.</DIV>
    <SCRIPT TYPE="text/javascript">
      var contObject = document.all["cont"];
      var receiver1Object = document.all["receiver1"];
      var receiver2Object = document.all["receiver2"];
      var piObject = jspsGetProgramInfo();
      if ((piObject.programName == JSPS_GPI_MSIE) ||
      ↳(piObject.programName == JSPS_GPI_OPERA))
        {
          contObject.onmousedown = contOnMouseDownIEOpera;
          document.body.onmousemove = bodyOnMouseMoveIEOpera;
        }
      else
        {
          contObject.addEventListener("mousedown", contOnMouseDownFF,
          ↳false);
          document.body.addEventListener("mousemove", bodyOnMouseMoveFF,
          ↳false);
        }
      contObject.onmouseup = contOnMouseUp;
    </SCRIPT>
  </BODY>
</HTML>
```

Что дальше?

О свободно позиционируемых элементах можно писать много — слишком мощная это штука. Но мы, пожалуй, закончим, иначе так и не доберемся до других глав.

Следующая глава будет посвящена мультимедийным элементам. Мы узнаем, как поместить на Web-страницу мультимедийный элемент (звук, видео, графику Shockwave/Flash) и как управлять его воспроизведением из Web-сценариев. А также мы узнаем, как определить, установлен ли на компьютере посетителя нужный для просмотра мультимедийного содержимого модуль расширения Web-обозревателя или элемент ActiveX. Так что не будет терять времени.

ГЛАВА 11



Создание мультимедийных элементов и управление ими

Мультимедийные элементы Web-страниц — это аудио- и видеоклипы в различных форматах и графика Shockwave/Flash. Также к мультимедийным элементам можно отнести фильтры и преобразования — весьма интересные и зачастую полезные возможности Internet Explorer, которые почему-то мало кто применяет. В этой главе народ ополчится на них, а мы посмотрим, кто победит.

Работа с мультимедийными элементами

Начнет народ свои советы, конечно, с "классических" мультимедийных элементов страниц, к которым относятся аудио- и видеоклипы и графика Shockwave/Flash. Мы научимся у народа помещать их на Web-страницы и управлять ими.

Как поместить на страницу мультимедийный элемент?

Проблема

Мне нужно поместить на мою Web-страницу мультимедийный элемент (аудио, видеоклип или изображение Shockwave/Flash). Как мне это сделать?

Решение 1

Как мы знаем, для помещения на Web-страницу мультимедийного элемента можно использовать модули расширения Web-обозревателя или элементы ActiveX — особые программы, работающие совместно с Web-обозревателем. С точки зрения посетителя сайта они одинаковы, но по-разному взаимодействуют с Web-обозревателем и по-разному им поддерживаются.

Сначала мы выясним, как использовать для вывода мультимедийного элемента модуль расширения Web-обозревателя. Это более универсальное решение, т. к. модули расширения поддерживаются всеми Web-обозревателями, в отличие от элементов ActiveX, нормально поддерживаемых только Internet Explorer.

Для воспроизведения аудио- и видеоклипов будет использован один из проигрывателей мультимедийных файлов, установленных на компьютере. Это может быть проигрыватель Windows Media, поставляемый в составе Windows, Apple QuickTime или RealPlayer.

Народ предупреждает!

Проблема в том, что разные Web-обозреватели используют различные программы — проигрыватели мультимедийных файлов. И поэтому зачастую трудно предугадать, как один и тот же мультимедийный файл воспроизведется в Internet Explorer, Opera и Firefox (и воспроизведется ли вообще). В этом смысле меньше всего проблем с Internet Explorer — он воспроизводит все, что ему "подсунут", и использует для этого подходящий проигрыватель мультимедийных файлов (так, файлы avi воспроизводятся проигрывателем Windows Media, файлы mov — проигрывателем Apple QuickTime и т. д.).

Поэтому Web-страницами с мультимедийными элементами лучше не увлекаться. Если же без них никак не обойтись, то следует прямо на Web-странице с мультимедийным элементом предупредить посетителей, что лучше всего просматривать ее в Internet Explorer. Благо он установлен на всех современных компьютерах, работающих под управлением Windows.

Просмотр графики Shockwave/Flash осуществляется с помощью проигрывателя Shockwave/Flash, который также поставляется в составе всех современных Web-обозревателей. Единственное — для вывода какого-либо изображения Shockwave/Flash может потребоваться более "свежая" версия этого проигрывателя, и тогда посетителю придется его найти и установить на свой компьютер.

Народ советует

Вот, кстати, неплохой способ решения описанной выше проблемы несовместимости Web-обозревателей и проигрывателей мультимедийных файлов. Достаточно преобразовать нужный видеоклип в формат Shockwave/Flash — и прощай, несовместимость!

Итак, модули расширения Web-обозревателя. Давайте же выясним, как поместить их на Web-страницу.

Из всего богатства HTML нам понадобится лишь один тег. Это парный тег `<EMBED>`, служащий для помещения на Web-страницу модуля расширения, с помощью которого посетитель будет просматривать или прослушивать наш мультимедийный элемент. Формат его написания очень прост:

```
<EMBED SRC="<Интернет-адрес файла с данными>"
↳[PLUGINSPAGE="<Интернет-адрес Web-страницы с дистрибутивом модуля
↳расширения>"]
<Атрибуты тега, задающие прочие параметры модуля расширения>>
</EMBED>
```

Самый важный атрибут этого тега — `SRC`. Он задает интернет-адрес файла с данными (аудио, видеоклипа или изображения Shockwave/Flash).

Необязательный атрибут `PLUGINSPAGE` задает интернет-адрес Web-страницы, где посетитель может найти дистрибутив модуля расширения. Это будет очень полезно, если на компьютере посетителя данный модуль расширения не установлен. В частности, для проигрывателя Shockwave/Flash значение этого атрибута будет таким:

http://www.macromedia.com/go/getflashplayer

Тег `<EMBED>` поддерживает множество других атрибутов, задающих дополнительные параметры модуля расширения. Эти параметры у каждого модуля расширения разные; полный их список должен быть приведен в документации к модулю расширения на сайте его разработчика.

Также тег `<EMBED>` поддерживает атрибут `ID`, с помощью которого задается имя мультимедийного элемента. Ну, это нам давно знакомо...

> ***Народ замечает***
>
> Вообще-то, тег `<EMBED>` также поддерживает атрибуты `WIDTH` и `HEIGHT`, задающие, соответственно, ширину и высоту модуля расширения. Однако того же самого эффекта можно достичь с помощью стиля CSS (атрибуты `width` и `height`). Наверно, поэтому автор книги не упомянул об этих атрибутах.

Пример 1

Вот HTML-код Web-страницы, содержащей аудиоклип:

```
<HTML>
  <HEAD>
    <TITLE>Музыка на Web-странице</TITLE>
  </HEAD>
  <BODY>
    <EMBED SRC="sound.wav">
    </EMBED>
  </BODY>
</HTML>
```

Здесь мы задали только интернет-адрес файла с данными (значение `"sound.wav"` атрибута `SRC` тега `<EMBED>`). И этого вполне достаточно.

Если мы откроем эту страницу в Internet Explorer, то видеоклип будет сразу же загружен и воспроизведен. Причем прямо на Web-странице будет присутствовать интерфейс выбранного Web-обозревателем проигрывателя мультимедийных файлов — бегунок и набор кнопок для управления воспроизведением.

Пример 2

А вот HTML-код Web-страницы, содержащей рекламный баннер в формате Shockwave/Flash:

```
<HTML>
  <HEAD>
    <TITLE>Страница с баннером</TITLE>
  </HEAD>
  <BODY>
    <EMBED SRC="banner.swf">
    </EMBED>
  </BODY>
</HTML>
```

> ***Народ предупреждает!***
>
> Мультимедийные элементы на Web-страницах следует использовать только в том случае, если они там действительно нужны, т. е. являются частью содержимого страниц. Просто для красоты их применять не рекомендуется: как правило, они очень долго загружаются, отвлекают посетителя и не дают ему никакой особо полезной информации.

Решение 2 (Internet Explorer)

Для вывода мультимедийных элементов также можно использовать элементы ActiveX. Этими элементами могут быть проигрыватели мультимедийных файлов (в частности, проигрыватель Windows Media) и проигрыватель Shockwave/Flash.

С одной стороны, элементы ActiveX не столь "совместимы", как модули расширения Web-обозревателя — они поддерживаются только Internet Explorer. С другой же, с ними меньше проблем, т. к. мы можем однозначно указать, с помощью какого элемента ActiveX должен воспроизводиться наш мультимедийный элемент (что-то вроде "возьми проигрыватель Windows Media и крути в нем этот видеоклип"). С модулями расширения такой фокус не проходит — Web-обозреватель сам подбирает наиболее подходящий, с его точки зрения, модуль, исходя из собственных настроек.

Чтобы поместить на страницу элемент ActiveX, нам понадобятся уже два тега. Первый — парный тег `<OBJECT>`, служащий собственно для помещения на

Web-страницу элемента ActiveX, с помощью которого посетитель будет просматривать или прослушивать наш мультимедийный элемент. Второй — <PARAM>, задающий дополнительные параметры для данного элемента ActiveX (интернет-адрес файла с данными, необходимость наличия интерфейса управления и пр.).

Формат записи парного тега <OBJECT> таков:

```
<OBJECT CLASSID="<GUID элемента ActiveX>"
↳[CODEBASE="<Интернет-адрес дистрибутива элемента ActiveX>"]>
  <Теги <PARAM>, задающие параметры элемента ActiveX>
</OBJECT>
```

В теге <OBJECT> могут присутствовать и другие атрибуты, например, атрибут ID, с помощью которого можно задать имя этого элемента страницы, чтобы обращаться к нему из сценариев. Но приведенных выше двух атрибутов будет вполне достаточно для его работы.

Народ замечает

Тег <OBJECT> также поддерживает атрибуты WIDTH и HEIGHT.

Атрибут CLASSID очень важен. Он задает так называемый *GUID* (Global Unique IDentifier, глобальный уникальный идентификатор) элемента ActiveX, однозначно его идентифицирующий. С помощью GUID мы даем Web-обозревателю понять, какой элемент ActiveX хотим использовать для создания мультимедийного элемента.

Но как узнать нужный GUID?! Очень просто. GUID всех установленных в системе элементов ActiveX записывается в "ветви"

```
HKEY_CLASSES_ROOT/CLSID
```

системного реестра. Web-обозреватель, встретив в HTML-коде страницы GUID, отыскивает его в указанной "ветви" реестра, выясняет, какой элемент ActiveX нужно использовать, загружает его и запускает на выполнение.

Народ советует

Чтобы найти в указанной "ветви" системного реестра нужный GUID, лучше всего воспользоваться встроенной в программу Редактор реестра возможностью поиска.

Так, стандартно поставляемый в составе Windows проигрыватель Window Media (в смысле, соответствующий ему элемент ActiveX) имеет вот такой GUID:

```
22D6F312-B0F6-11D0-94AB-0080C74C7E95
```

Что касается проигрывателя Shockwave/Flash, то ему соответствует вот такой GUID:

```
d27cdb6e-ae6d-11cf-96b8-444553540000
```

А теперь — внимание! GUID в коде HTML должен быть записан таким образом:

```
clsid:<Нужный нам GUID>
```

Символы "clsid:" являются обязательными. Пробелы между двоеточием и GUID не допускаются.

Атрибут `CODEBASE` тега `<OBJECT>` является необязательным и задает интернет-адрес дистрибутива данного элемента ActiveX. Если данный элемент ActiveX не установлен на компьютере клиента, сам Web-обозреватель загрузит его дистрибутив, пользуясь этим адресом, и сам же его установит.

Для проигрывателя Windows Media атрибут `CODEBASE` указывать не нужно — он и так стандартно поставляется в составе Windows, начиная с версии 98. А вот для проигрывателя Shockwave/Flash его лучше указать; его значение должно быть таким:

http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=8,0,0,0

Вдруг для просмотра какого-либо изображения понадобится более новая версия этого проигрывателя!

Пожалуй, о теге `<OBJECT>` больше рассказывать нечего. Обратимся к тегу `<PARAM>`.

Одинарный тег `<PARAM>` должен присутствовать только внутри тега `<OBJECT>`. Он служит для задания дополнительных параметров модуля расширения. Вот формат его записи:

```
<PARAM NAME="<Имя параметра>" VALUE="<Значение параметра>">
```

Думается, комментировать здесь нечего.

По крайней мере, один тег `<PARAM>` всегда присутствует в теге `<OBJECT>` — он задает интернет-адрес файла с данными. В случае проигрывателя Windows Media за это "отвечает" параметр `FileName`; проигрыватель Shockwave/Flash для этого использует параметр `movie`. В качестве значения обоих этих параметров указывается интернет-адрес файла данных.

> ***Народ предупреждает!***
>
> Обратим внимание — в теге `<EMBED>` для задания интернет-адреса файла с данными используется универсальный атрибут `SRC`, а в теге `<OBJECT>` его нет. В теге `<OBJECT>` файл с данными (как и все остальные дополнительные параметры) указывается с помощью тега `<PARAM>`.

Пример 1

Вот HTML-код Web-страницы, содержащей видеоклип:

```
<HTML>
  <HEAD>
    <TITLE>Кино на Web-странице</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
      <PARAM NAME="FileName" VALUE="clip.avi">
    </OBJECT>
  </BODY>
</HTML>
```

Пример 2

А вот HTML-код Web-страницы, содержащей рекламный баннер в формате Shockwave/Flash:

```
<HTML>
  <HEAD>
    <TITLE>Страница с баннером</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000">
      <PARAM NAME="movie" VALUE="banner.swf">
    </OBJECT>
  </BODY>
</HTML>
```

Народ советует

Очень часто теги <EMBED> и <OBJECT> объединяют, помещая первый внутрь последнего. Вот так:

```
<OBJECT CLASSID="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000">
  <PARAM NAME="movie" VALUE="banner.swf">
  <EMBED SRC="banner.swf">
  </EMBED>
</OBJECT>
```

Тогда Web-обозреватели, поддерживающие тег <OBJECT>, прочитают этот тег и проигнорируют тег <EMBED>. А не поддерживающие тег <OBJECT> Web-обозреватели, наоборот, проигнорируют его, но прочитают тег <EMBED>. Это называется — угодить всем.

Решение 3

В принципе, для этого мы можем использовать обычную гиперссылку, указывающую на файл с мультимедийными данными. Тогда этот файл будет открыт в программе, зарегистрированной для данного типа файлов по умолчанию, т. е. в проигрывателе мультимедийных файлов.

Пожалуй, это самый простой способ поместить на Web-страницу мультимедийный элемент. Однако он имеет большой недостаток — мы не можем контролировать параметры программы, в которой будет открыт этот файл, и даже не сможем узнать, в какой именно программе он будет открыт. Так, если на компьютере посетителя установлена программа массовой загрузки файлов (ReGet, FlashGet, Download Master или др.), мультимедийный файл может быть перехвачен этой программой и помещен в список загружаемых файлов. И посетитель сможет просмотреть его только тогда, когда он полностью загрузится.

Народ замечает

А так ли уж нужно, чтобы мультимедийный файл воспроизводился прямо на Web-странице? Может, лучше просто поместить на страницу ссылку на него и дать возможность посетителю просто загрузить его и просмотреть, как и когда ему захочется?..

Параметры мультимедийного элемента

Проблема

Мне очень нужно задать некоторые параметры модуля расширения Web-обозревателя или элемента ActiveX, с помощью которого воспроизводится мультимедийный элемент страницы. Можно ли узнать список этих параметров?

Решение

Можно-то можно, только вот различных модулей расширения Web-обозревателя и элементов ActiveX существует очень много, и список всех их параметров займет всю эту книгу. Пожалуй, стоит перечислить только параметры проигрывателя Windows Media и проигрывателя Shockwave/Flash, как наиболее часто используемых.

Проигрыватель Windows Media поддерживает довольно много параметров. В табл. 11.1 приведены, по мнению автора книги, наиболее полезные из них. Все эти параметры задаются либо атрибутами тега `<EMBED>`, либо тегами `<PARAM>`.

Набор параметров, поддерживаемых проигрывателем Shockwave/Flash, не столь впечатляет. Некоторые из них перечислены в табл. 11.2. Они также задаются либо атрибутами тега `<EMBED>`, либо тегами `<PARAM>`.

Таблица 11.1. *Некоторые параметры проигрывателя Windows Media, задаваемые атрибутами тега* <EMBED> *и тегами* <PARAM>

Параметр	Описание
AutoRewind	Если "1", воспроизводимый мультимедийный файл будет автоматически перемотан к началу после окончания воспроизведения. Если "0", файл не будет перематываться. Значение по умолчанию — "1"
AutoSize	Если "1", изображение в видеоклипе будет подстраиваться под размер проигрывателя, если "0", то не будет. Если размеры проигрывателя заданы тегами WIDTH и HEIGHT или одноименными атрибутами стиля, значение по умолчанию — "1", иначе — "0". Действует только в проигрывателе Windows Media 6.4
AutoStart	Если "1", воспроизведение мультимедийного файла будет запущено сразу же после его загрузки. Если "0", то посетитель сам должен запустить его воспроизведение. Значение по умолчанию — "1"
Balance	Задает величину стереобаланса. Принимает значения от –10 000 до 10 000. Значение по умолчанию — 0
EnableContextMenu	Если "1", после щелчка правой кнопкой мыши на проигрывателе появится контекстное меню, если "0", контекстное меню появляться не будет. Значение по умолчанию — "1"
Enabled	Если "1", посетитель может управлять проигрывателем с помощью кнопок, бегунка и контекстного меню, если "0", то не может. Значение по умолчанию — "1"
EnablePositionControls	Если "1", будут доступны кнопки перемотки, если "0", то не будут доступны. Значение по умолчанию — "1"
EnableTrackbar	Если "1", будет доступен бегунок, если "0", то не будет доступен. Значение по умолчанию — "1"
FileName	Задает интернет-адрес мультимедийного файла. Используется только в тегах <PARAM>; в теге <EMBED> следует применять атрибут SRC
Mute	Если "1", звук будет приглушен, если "0", звук будет воспроизводиться с изначальной громкостью. Значение по умолчанию — "0"
PlayCount	Задает количество повторов воспроизведения мультимедийного файла. Если 0, файл будет воспроизводиться бесконечно. Значение по умолчанию — 1 (т. е. однократное воспроизведение файла)
Rate	Задает относительную частоту кадров воспроизводимого мультимедийного файла в виде множителя, на который нужно умножить изначальную частоту кадров. Например, если задать значение .5, то файл будет воспроизводиться с замедленной в два раза скоростью. Значение по умолчанию — 1.0

Таблица 11.1 *(продолжение)*

Параметр	Описание
SendErrorEvents	Если "1", проигрыватель будет генерировать события Error, если "0", не будет генерировать. Значение по умолчанию — "1"
SendKeyboardEvents	Если "1", проигрыватель будет генерировать события клавиатуры, если "0", не будет генерировать. Значение по умолчанию — "0"
SendMouseClickEvents	Если "1", проигрыватель будет генерировать события Click, DblClick, MouseDown и MouseUp, если "0", не будет генерировать. Значение по умолчанию — "0"
SendMouseMoveEvents	Если "1", проигрыватель будет генерировать события MouseMove, если "0", не будет генерировать. Значение по умолчанию — "0"
SendOpenStateChangeEvents	Если "1", проигрыватель будет генерировать события OpenStateChange, если "0", не будет генерировать. Значение по умолчанию — "1"
SendPlayStateChangeEvents	Если "1", проигрыватель будет генерировать события PlayStateChange, если "0", не будет генерировать. Значение по умолчанию — "1"
SendWarningEvents	Если "1", проигрыватель будет генерировать события Warning, если "0", не будет генерировать. Значение по умолчанию — "1"
ShowAudioControls	Если "1", проигрыватель выведет на экран органы управления звуковым сопровождением, если "0", не выведет. Значение по умолчанию — "1"
ShowControls	Если "1", проигрыватель выведет на экран кнопки запуска, приостановки и остановки воспроизведения, если "0", не выведет. Значение по умолчанию — "1"
ShowDisplay	Если "1", проигрыватель выведет на экран область просмотра видео, если "0", не выведет. Значение по умолчанию — "0", если воспроизводится аудиоклип, и "1", если воспроизводится видеоклип
ShowPositionControls	Если "1", проигрыватель выведет на экран кнопки перемотки, если "0", то недоступны. Значение по умолчанию — "1"
ShowStatusBar	Если "1", проигрыватель выведет на экран строку статуса, если "0", не выведет. Значение по умолчанию — "0"
ShowTracker	Если "1", проигрыватель выведет на экран бегунок, если "0", не выведет. Значение по умолчанию — "1"
TransparentAtStart	Если "1", проигрыватель будет прозрачным перед началом и после окончания воспроизведения мультимедийного файла, если "0", не будет. Значение по умолчанию — "0"

Таблица 11.1 *(окончание)*

Параметр	Описание
Volume	Задает громкость звука в диапазоне от –10 000 (минимальная) до 0 (максимальная). Значение по умолчанию — –600
WindowlessVideo	Если "1", видео будет выводиться прямо на Web-странице (безоконный режим), если "0", видео будет выводиться в отдельном "окне", располагаемым на этой странице. Безоконный режим следует включить, если к воспроизводимому видео предполагается применять какие-либо эффекты. Значение по умолчанию — "0"

Таблица 11.2. *Некоторые параметры проигрывателя Shockwave/Flash, задаваемые атрибутами тега* <EMBED> *и тегами* <PARAM>

Параметр	Описание
align	Задает выравнивание "окна" проигрывателя на Web-странице. Доступны значения: "Default" (выравнивание по центру страницы), "L" (по левой границе), "T" (по верхней), "R" (по правой) и "B" (по нижней)
base	Задает базовый интернет-адрес, используемый для расчета полных интернет-адресов по относительным
loop	Если "true", фильм будет зациклен, если "false", не будет. Значение по умолчанию — "true"
menu	Если "true", при щелчке мышью на проигрывателе будет выведено полное контекстное меню, если "false", это меню будет содержать только пункт **About Flash Player**, выводящий диалоговое окно сведений о проигрывателе. Значение по умолчанию — "true"
movie	Задает интернет-адрес файла с изображением Shockwave/Flash
play	Если "true", воспроизведение фильма будет начато сразу же после его загрузки, если "false", посетитель сам должен запустить его воспроизведение. Значение по умолчанию — "true"
quality	Задает качество воспроизведения графики. Может принимать значения: "low" (самое низкое качество), "autolow", "autohigh", "medium", "high" и "best" (самое высокое качество). Значение по умолчанию — "high"
salign	Задает выравнивание изображения в "окне" проигрывателя. Доступны значения: "L" (по левой границе "окна"), "T" (по верхней), "R" (по правой), "B" (по нижней), "TL" (по верхней и левой), "TR" (по верхней и правой), "BL" (по нижней и левой) и "BR" (по нижней и правой). Если не задан, изображение располагается в центре "окна" проигрывателя

Таблица 11.2 *(окончание)*

Параметр	Описание
scale	Задает параметры масштабирования изображения в "окне" проигрывателя. Доступны значения: "showall" (масштабирование с сохранением пропорций, но могут появиться пустые области), "noborder" (то же самое, но без пустых областей — проигрыватель обрежет изображение, чтобы не допустить их появления) и "exactfit" (масштабирование без сохранения пропорций). Значение по умолчанию — "showall"
swliveconnect	Если "1", то Web-обозреватель загрузит исполняющую среду JavaScript, и тогда станет возможно управлять проигрывателем из Web-сценариев. Если "0", Web-обозреватель не будет этого делать. Значение по умолчанию — "0". Применяется только в теге <EMBED>
wmode	Задает параметры вывода изображения. Доступны значения "Window" (вывод изображения в собственном "окне" проигрывателя, которое будет располагаться на Web-странице), "Opaque" (вывод изображения прямо на странице, причем элементы страницы, располагающиеся под изображением, не будут видны) и "Transparent" (то же самое, но элементы страницы, располагающиеся под изображением, будут видны). Значение по умолчанию — "Window"

Народ советует

Полный список параметров проигрывателя Windows Media можно узнать на посвященной ему странице сайта MSDN (**http://msdn.microsoft.com/library/en-us/wmp6sdk/htm/microsoftwindowsmediaplayercontrolversion64sdk.asp**).
Полный список параметров проигрывателя Shockwave/Flash можно найти в интерактивной справке Flash или на сайте фирмы Adobe (**http://www.adobe.com**) в разделе, посвященном Flash.

Пример 1

Далее приведен HTML-код Web-страницы, содержащей бесконечно воспроизводящийся фоновый звук.

```
<HTML>
  <HEAD>
    <TITLE>Эта музыка будет вечной</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
      <PARAM NAME="FileName" VALUE="sound.wav">
      <PARAM NAME="PlayCount" VALUE="0">
      <PARAM NAME="ShowAudioControls" VALUE="0">
      <PARAM NAME="ShowControls" VALUE="0">
      <PARAM NAME="ShowDisplay" VALUE="0">
```

```
      <PARAM NAME="ShowPositionControls" VALUE="0">
      <PARAM NAME="ShowStatusBar" VALUE="0">
      <PARAM NAME="ShowTracker" VALUE="0">
    </OBJECT>
  </BODY>
</HTML>
```

Заметим, какие параметры мы задали для проигрывателя Windows Media. Во-первых, мы задали бесконечное повторение воспроизведения звукового клипа (значение 0 параметра `PlayCount`). Во-вторых, мы скрыли все элементы управления проигрывателя (значения 0 для параметров `ShowAudioControls`, `ShowControls`, `ShowDisplay`, `ShowPositionControls`, `ShowStatusBar` и `ShowTracker`). После этого невидимый проигрыватель будет постоянно играть один и тот же звук, действуя посетителям на нервы...

Народ советует

Автор книги, кстати, дело говорит. Ничто так не действует на нервы, как бесконечно пиликающая фоновая музыка. Пожалейте своих посетителей, не "озвучивайте" ваши страницы таким образом!

Пример 2

А вот HTML-код Web-страницы с баннером Shockwave/Flash, растянутом по горизонтали.

```
<HTML>
  <HEAD>
    <TITLE>Баннер</TITLE>
  </HEAD>
  <BODY>
    <EMBED SRC="banner.swf" SCALE="exactfit" STYLE="width: 400px">
    </EMBED>
  </BODY>
</HTML>
```

Здесь мы задали для "окна" проигрывателя Shockwave/Flash ширину, равную 400 пикселам, с помощью атрибута стиля `width`. После этого мы задали режим масштабирования изображения Shockwave/Flash без сохранения пропорций (значение `"exactfit"` параметра `scale` — он задается с помощью одноименного атрибута тега `<EMBED>`).

Народ советует

Разумеется, автор книги исказил пропорции изображения только для примера. С реальными изображениями такого делать не стоит.

Свойства мультимедийного элемента

Проблема

Хорошо, с параметрами все понятно. А как насчет свойств мультимедийного элемента?

Решение

Свойства проигрывателя Windows Media весьма многочисленны. С большинством из них мы уже знакомы — см. табл. 11.1. Да, к перечисленным там параметрам мы можем обращаться и как к свойствам. Единственное — вместо значений "1" и "0" следует использовать логические значения `true` и `false`.

Кроме того, проигрыватель Windows Media поддерживает и другие свойства, перечисленные в табл. 11.3. Большинство из них доступно только для чтения и возвращает различную служебную информацию, которую можно использовать в сценариях. Хотя некоторые свойства доступны и для записи и позволяют задавать какие-либо параметры проигрывателя.

Таблица 11.3. *Некоторые свойства, поддерживаемые проигрывателем Windows Media*

Свойство	Описание
`Bandwidth`	Возвращает скорость загрузки мультимедийного файла в битах в секунду в числовом виде
`BufferingCount`	Возвращает в числовом виде, сколько раз во время воспроизведения мультимедийного файла выполнялась его *буферизация* (предварительная загрузка в особую область памяти, называемую *буфером*)
`BufferingProgress`	Возвращает процент заполнения буфера в числовом виде
`BufferingTime`	Возвращает длительность в секундах фрагмента мультимедийного файла, загруженного в буфер. Значение возвращается в числовом виде
`CurrentPosition`	Задает текущую позицию бегунка в секундах в числовом виде
`Duration`	Возвращает продолжительность мультимедийного файла в секундах в числовом виде
`ErrorCode`	Возвращает в числовом виде код возникшей ошибки
`ErrorDescription`	Возвращает в строковом виде описание возникшей ошибки
`HasError`	Возвращает `true`, если в процессе загрузки или воспроизведения мультимедийного файла возникла ошибка
`ImageSourceHeight`	Возвращает высоту изображения видеоклипа в пикселах в числовом виде

Таблица 11.3 *(окончание)*

Свойство	Описание
`ImageSourceWidth`	Возвращает ширину изображения видеоклипа в пикселах в числовом виде
`IsBroadcast`	Возвращает `true`, если выполняется прием потокового аудио или видео
`IsDurationValid`	Возвращает `true`, если проигрыватель может правильно определить продолжительность аудио- или видеоклипа. Это свойство часто бывает нужно, т. к. при приеме потокового аудио или видео проигрыватель не всегда может определить его продолжительность
`LostPackets`	Возвращает количество потерянных *пакетов* (отдельных фрагментов данных) потокового аудио или видео в числовом виде
`OpenState`	Возвращает состояние открытия мультимедийного файла. Возвращаемые свойством значения: 0 (файл не открыт), 1 (идет загрузка файла описания потокового аудио или видео), 2 (идет загрузка файла описания интернет-радиостанции), 3 (поиск сервера), 4 (соединение с сервером), 5 (открытие файла) и 6 (файл открыт)
`PlayState`	Возвращает состояние воспроизведения мультимедийного файла. Возвращаемые свойством значения: 0 (воспроизведение остановлено), 1 (воспроизведение приостановлено), 2 (файл воспроизводится), 3 (идет буферизация потока), 4 (идет прокрутка вперед), 5 (идет прокрутка назад), 6 (выполняется переход на следующую позицию), 7 (выполняется переход на предыдущую позицию) и 8 (файл не был открыт)
`ReadyState`	Возвращает состояние готовности проигрывателя к воспроизведению мультимедийного файла. Возвращаемые свойством значения: 0 (файл не был задан, т. е. свойству `FileName` не был присвоен его интернет-адрес), 1 (идет загрузка файла), 3 (загружен не весь файл, но воспроизведение можно начинать), 4 (файл загружен полностью)
`ReceivedPackets`	Возвращает количество принятых пакетов потокового аудио или видео в числовом виде
`ReceptionQuality`	Возвращает процент принятых пакетов от общего количества пакетов потокового аудио или видео, пришедших за последние 30 секунд. Значение возвращается в числовом виде
`RecoveredPackets`	Возвращает количество восстановленных пакетов потокового аудио или видео в числовом виде

Что касается проигрывателя Shockwave/Flash, то доступ к его свойствам можно получить только в том случае, если он реализован в виде элемента ActiveX. Все эти свойства перечислены в табл. 11.4.

Таблица 11.4. *Свойства, поддерживаемые проигрывателем Shockwave/Flash, реализованным в виде элемента ActiveX*

Свойство	Описание
`AlignMode`	Задает выравнивание "окна" проигрывателя на Web-странице. Доступны значения 0 (выравнивание по центру страницы), 1 (по левой границе) , 2 (по правой), 4 (по верхней) и 8 (по нижней)
`FrameNum`	Задает номер текущего кадра фильма
`IsPlaying`	Возвращает `true`, если фильм в данный момент воспроизводится
`Loop`	Если `true`, фильм будет зациклен, если `false`, не будет
`Movie`	Задает интернет-адрес файла с изображением Shockwave/Flash
`Quality`	Задает качество вывода изображения. Доступны значения от 0 (самое низкое качество) до 3 (самое высокое качество)
`ReadyState`	Возвращает состояние, в котором находится изображение Shockwave/Flash. Значения этого свойства: 0 (изображение только начало загружаться), 1 (часть изображения загружена, но воспроизведение еще не началось), 2 (часть изображения загружена, и скоро начнется воспроизведение), 3 (часть изображения загружена, и изображение воспроизводится) и 4 (изображение полностью загружено)
`ScaleMode`	Задает параметры масштабирования изображения в "окне" проигрывателя. Доступны значения: 0 (масштабирование с сохранением пропорций, но могут появиться пустые области), 1 (то же самое, но без пустых областей — проигрыватель обрежет изображение, чтобы не допустить их появления) и 2 (масштабирование без сохранения пропорций)
`TotalFrames`	Возвращает количество кадров фильма в числовом виде

Народ советует

Полный список свойств проигрывателя Windows Media можно узнать на посвященной ему странице сайта MSDN (**http://msdn.microsoft.com/library/en-us/wmp6sdk/htm/microsoftwindowsmediaplayercontrolversion64sdk.asp**).

Полный список свойств проигрывателя Shockwave/Flash можно найти в разделе поддержки (**http://www.macromedia.com/support/flash**) Flash сайта фирмы Adobe.

Пример 1

Далее приведен HTML-код Web-страницы, выводящей на экран продолжительность воспроизводимого в проигрывателе Windows Media аудиоклипа.

```
<HTML>
  <HEAD>
    <TITLE>Музыка на Web-странице</TITLE>
  </HEAD>
```

```
  <BODY>
    <OBJECT CLASSID="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
    ↳ID="acx">
      <PARAM NAME="FileName" VALUE="sound.wav">
    </OBJECT>
    <P ID="output">Продолжительность аудиоклипа:</P>
    <SCRIPT TYPE="text/javascript">
      function writeDate()
        {
          var acxObject = document.all["acx"];
          var outputObject = document.all["output"];
          outputObject.firstChild.nodeValue += " " +
          ↳acxObject.Duration.toString() + " секунд.";
        }

      window.setTimeout(writeDate, 1000);
    </SCRIPT>
  </BODY>
</HTML>
```

Но почему мы поместили код, выводящий эту самую продолжительность на страницу, в функцию и вызвали ее с секундной задержкой? (Метод `setTimeout` объекта `window`, как мы помним, как раз и задает временную задержку перед вызовом переданной ему первым параметром функции.) Дело в том, что Web-обозревателю требуется время на загрузку клипа. А пока клип не загружен, свойство `Duration` проигрывателя Windows Media всегда возвращает 0.

Пример 2

А вот HTML-код Web-страницы с баннером в формате Shockwave/Flash, выводящей на экран количество кадров в нем.

```
<HTML>
  <HEAD>
    <TITLE>Баннер</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    ↳ID="acx">
      <PARAM NAME="movie" VALUE="banner.swf">
    </OBJECT>
    <P ID="output">Количество кадров в баннере:</P>
    <SCRIPT TYPE="text/javascript">
      function writeDate()
```

```
        {
          var acxObject = document.all["acx"];
          var outputObject = document.all["output"];
          outputObject.firstChild.nodeValue += " " +
          ↳acxObject.TotalFrames.toString() + ".";
        }

      window.setTimeout(writeDate, 1000);
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь мы тоже задаем секундную задержку перед определением количества кадров в баннере, дабы дождаться, когда он загрузится полностью.

Методы мультимедийного элемента

Проблема

Ну и для полноты картины неплохо было бы узнать, какие методы поддерживают мультимедийные элементы.

Решение

Наиболее полезные в Web-программировании методы проигрывателя Windows Media перечислены в табл. 11.5.

Таблица 11.5. *Некоторые методы, поддерживаемые проигрывателем Windows Media*

Метод	Описание
`FastForward()`	Выполняет быструю прокрутку воспроизводимого файла вперед
`FastReverse()`	Выполняет быструю прокрутку воспроизводимого файла назад
`Open(<`*Интернет-адрес файла*`>)`	Открывает мультимедийный файл, интернет-адрес которого был передан в качестве единственного параметра
`Pause()`	Приостанавливает воспроизведение мультимедийного файла
`Play()`	Запускает воспроизведение мультимедийного файла
`Stop()`	Останавливает воспроизведение мультимедийного файла

Что касается проигрывателя Shockwave/Flash, то он более богат на полезные для нас методы (табл. 11.6).

Таблица 11.6. *Некоторые методы, поддерживаемые проигрывателем Shockwave/Flash*

Метод	Описание
`GetVariable(<`*Путь переменной*`>)`	Возвращает значение переменной с заданным в строковом виде *Путем*. Если переменная не существует, возвращает `null`
`GotoFrame(<`*Номер кадра*`>)`	Перемещает указатель на кадр с заданным в числовом виде номером. Нумерация кадров начинается с нуля
`IsPlaying()`	Возвращает `true`, если фильм в данный момент воспроизводится
`LoadMovie(<`*Номер слоя*`>, <`*Интернет-адрес файла*`>)`	Загружает файл с изображением Shockwave/Flash, интернет-адрес которого был задан вторым параметром, в слой (о слоях см. интерактивную документацию Flash), номер которого задан первым параметром. Первый параметр должен быть задан в числовом виде, второй — в строковом
`Pan(<`*X*`>, <`*Y*`>, <`*Режим*`>)`	Сдвигает изображение, увеличенное методом `SetZoomRect`, на *X* по горизонтали и на *Y* по вертикали. Если *Режим* равен 0, то *X* и *Y* измеряются в пикселах, если 1, то в процентах от размера "окна" проигрывателя Flash. Все параметры должны быть заданы в числовом виде
`PercentLoaded()`	Возвращает, сколько процентов изображения Shockwave/Flash загружено к данному моменту, в числовом виде
`Play()`	Запускает воспроизведение фильма
`Rewind()`	Перематывает фильм на начало
`SetVariable(<`*Путь переменной*`>, <`*Значение*`>)`	Присваивает *Значение* переменной с заданным *Путем*. Оба параметра должны быть заданы в строковом виде
`SetZoomRect(<`*X1*`>, <`*Y1*`>, <`*X2*`>, <`*Y2*`>)`	Показывает в "окне" проигрывателя увеличенный фрагмент изображения так, чтобы он занял все "окно". Посетитель может перетаскивать фрагмент мышью, чтобы рассмотреть его целиком. *X1* и *Y1* — координаты левого верхнего угла, а *X2* и *Y2* — координаты правого нижнего угла увеличиваемого фрагмента. Координаты задаются в твипах; 20 твипов составляют один пиксел. Все параметры должны быть заданы в числовом виде
`StopPlay()`	Останавливает воспроизведение фильма
`TCurrentFrame(<`*Клип*`>)`	Возвращает номер текущего кадра в заданном *Клипе*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом `/`)

Таблица 11.6 *(окончание)*

Метод	Описание
`TCurrentLabel(<`*Клип*`>)`	Возвращает имя текущего кадра в заданном *Клипе*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом /). Если кадр не имеет имени, возвращается пустая строка
`TGetProperty(<`*Клип*`>, <`*Номер свойства*`>)`	Возвращает значение свойства *Клипа* с заданным *Номером*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом /), *Номер свойства* — в числовом виде. Номера всех доступных свойств перечислены в табл. 11.7
`TGotoFrame(<`*Клип*`>, <`*Номер кадра*`>)`	Выполняет переход на кадр с заданным *Номером* заданного *Клипа*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом /), *Номер кадра* — в числовом виде
`TGotoLabel(<`*Клип*`>, <`*Имя кадра*`>)`	Выполняет переход на кадр с заданным *Именем* заданного *Клипа*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом /), *Имя кадра* — также в строковом виде
`TotalFrames()`	Возвращает в числовом виде количество кадров в фильме
`TPlay(<`*Клип*`>)`	Запускает воспроизведение заданного *Клипа*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом /)
`TSetProperty(<`*Клип*`>, <`*Номер свойства*`>, <`*Значение*`>)`	Присваивает новое *Значение* свойству *Клипа* с заданным *Номером*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом /), *Номер свойства* — в числовом виде. Номера всех доступных свойств перечислены в табл. 11.7
`TStopPlay(<`*Клип*`>)`	Останавливает воспроизведение заданного *Клипа*. *Клип* задается в строковом виде в виде пути (так, основной фильм обозначается символом /)
`Zoom(<`*Масштаб*`>)`	Масштабирует изображение в "окне" проигрывателя. Величина масштаба вычисляется по формуле: `100 / `*Масштаб*. Параметр должен быть задан в числовом виде

Таблица 11.7. *Номера свойств, доступных через методы* `TGetProperty` *и* `TSetProperty` *проигрывателя Shockwave/Flash*

Номер свойства	Описание
0	Горизонтальная координата
1	Вертикальная координата

Таблица 11.7 *(окончание)*

Номер свойства	Описание
2	Масштаб по горизонтали
3	Масштаб по вертикали
4	Номер текущего кадра (только для чтения)
5	Количество кадров в фильме (только для чтения)
6	Уровень прозрачности
7	Видимость
8	Ширина (только для чтения)
9	Высота (только для чтения)
10	Угол поворота
11	Не удалось установить
12	Количество загруженных кадров (только для чтения)
13	Имя
14	Не удалось установить
15	Интернет-адрес файла с изображением Shockwave/Flash (только для чтения)

Народ предупреждает!

Изображение Shockwave/Flash должно быть создано таким образом, чтобы позволять управлять собой из Web-сценариев путем вызова методов. Как это сделать, описано в интерактивной документации Flash. По умолчанию же доступ из Web-сценариев к проигрывателю Shockwave/Flash сильно ограничен.

Все сказанное выше относится к проигрывателю Shockwave/Flash версии 8 и более новых. Старые версии проигрывателя не так щепетильны в вопросах безопасности.

Народ советует

Полный список методов проигрывателя Windows Media можно узнать на посвященной ему странице сайта MSDN (**http://msdn.microsoft.com/library/en-us/wmp6sdk/htm/microsoftwindowsmediaplayercontrolversion64sdk.asp**).
Полный список методов проигрывателя Shockwave/Flash можно найти в разделе поддержки (**http://www.macromedia.com/support/flash**) Flash сайта фирмы Adobe.

Пример

Далее приведен HTML-код Web-страницы с фоновым звуком, воспроизведение которого можно приостанавливать и запускать снова.

```
<HTML>
  <HEAD>
    <TITLE>Фоновый звук</TITLE>
    <SCRIPT TYPE="text/javascript">
      function doPause()
        {
          var asxObject = document.all["asx"];
          asxObject.Pause();
        }

      function doStart()
        {
          var asxObject = document.all["asx"];
          asxObject.Play();
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
    ↳ID="asx">
      <PARAM NAME="FileName" VALUE="sound.wav">
      <PARAM NAME="PlayCount" VALUE="0">
      <PARAM NAME="ShowAudioControls" VALUE="0">
      <PARAM NAME="ShowControls" VALUE="0">
      <PARAM NAME="ShowDisplay" VALUE="0">
      <PARAM NAME="ShowPositionControls" VALUE="0">
      <PARAM NAME="ShowStatusBar" VALUE="0">
      <PARAM NAME="ShowTracker" VALUE="0">
    </OBJECT>
    <FORM>
      <INPUT TYPE="button" VALUE="Пауза" ONCLICK="doPause();">
      <INPUT TYPE="button" VALUE="Пуск" ONCLICK="doStart();">
    </FORM>
  </BODY>
</HTML>
```

Здесь мы взяли написанную ранее Web-страницу с фоновым звуком и просто добавили кнопки управления и соответствующие сценарии.

События мультимедийного элемента

Проблема

А что насчет событий мультимедийного элемента? Можно ли их обрабатывать, и если можно, то как?

Решение

Те события проигрывателя Windows Media, что будут полезными Web-программистам, перечислены в табл. 11.8. Заметим, что в их именах отсутствуют привычные нам символы "on".

Таблица 11.8. *Некоторые события, поддерживаемые проигрывателем Windows Media*

Событие	Описание
`Buffering`	Возникает при начале или завершении буферизации данных. Единственный параметр, передаваемый обработчику, принимает значение `true`, если буферизация данных началась, и `false`, если она закончилась
`Click`	Возникает при щелчке мышью на проигрывателе. Обработчику передаются следующие параметры: обозначение нажатой кнопки мыши — сумма чисел 1 (левая кнопка), 2 (правая кнопка) и 4 (средняя кнопка); обозначение нажатой клавиши-модификатора — сумма чисел 1 (была нажата клавиша <Shift>), 2 (клавиша <Ctrl>) и 4 (клавиша <Alt); горизонтальная координата точки, в которой был выполнен щелчок мышью, и ее вертикальная координата (обе координаты отсчитываются относительно "окна" проигрывателя). Все параметры передаются в числовом виде. Это событие генерируется только в том случае, если свойству (параметру) `SendMouseClickEvents` проигрывателя присвоено значение `true` ("1")
`DblClick`	Возникает при двойном щелчке мышью на проигрывателе. Обработчику передаются следующие параметры: обозначение нажатой кнопки мыши — сумма чисел 1 (левая кнопка), 2 (правая кнопка) и 4 (средняя кнопка); обозначение нажатой клавиши-модификатора — сумма чисел 1 (была нажата клавиша <Shift>), 2 (клавиша <Ctrl>) и 4 (клавиша <Alt); горизонтальная координата точки, в которой был выполнен двойной щелчок мышью, и ее вертикальная координата (обе координаты отсчитываются относительно "окна" проигрывателя). Все параметры передаются в числовом виде. Это событие генерируется только в том случае, если свойству (параметру) `SendMouseClickEvents` проигрывателя присвоено значение `true` ("1")
`Disconnect`	Возникает при разрыве соединения с сервером. Единственным параметром обработчику передается числовое значение, обозначающее причину разрыва
`EndOfStream`	Возникает при окончании воспроизведения мультимедийного файла. Единственным параметром обработчику передается числовое значение, обозначающее состояние файла; если передано значение 0, значит, файл был воспроизведен без проблем
`Error`	Возникает, если проигрыватель столкнулся с проблемой при попытке воспроизвести мультимедийный файл

Таблица 11.8 *(продолжение)*

Событие	Описание
KeyDown	Возникает при нажатии клавиши, если проигрыватель был активен (имел фокус ввода). Обработчику передаются следующие параметры: виртуальный код нажатой клавиши; обозначение нажатой клавиши-модификатора — сумма чисел 1 (была нажата клавиша <Shift>), 2 (клавиша <Ctrl>) и 4 (клавиша <Alt). Это событие генерируется только в том случае, если свойству (параметру) SendKeyboardEvents проигрывателя присвоено значение true ("1")
KeyPress	Возникает при нажатии и отпускании клавиши, если проигрыватель был активен (имел фокус ввода). Обработчику передается единственный параметр — код ASCII нажатой клавиши. Это событие генерируется только в том случае, если свойству (параметру) SendKeyboardEvents проигрывателя присвоено значение true ("1")
KeyUp	Возникает при отпускании нажатой ранее клавиши, если проигрыватель был активен (имел фокус ввода). Обработчику передаются следующие параметры: виртуальный код нажатой клавиши; обозначение нажатой клавиши-модификатора — сумма чисел 1 (была нажата клавиша <Shift>), 2 (клавиша <Ctrl>) и 4 (клавиша <Alt). Это событие генерируется только в том случае, если свойству (параметру) SendKeyboardEvents проигрывателя присвоено значение true ("1")
MouseDown	Возникает при нажатии кнопки мыши, если ее курсор находится над проигрывателем. Обработчику передаются следующие параметры: обозначение нажатой кнопки мыши — сумма чисел 1 (левая кнопка), 2 (правая кнопка) и 4 (средняя кнопка); обозначение нажатой клавиши-модификатора — сумма чисел 1 (была нажата клавиша <Shift>), 2 (клавиша <Ctrl>) и 4 (клавиша <Alt); горизонтальная координата точки, в которой находился курсор мыши, и ее вертикальная координата (обе координаты отсчитываются относительно "окна" проигрывателя). Все параметры передаются в числовом виде. Это событие генерируется только в том случае, если свойству (параметру) SendMouseClickEvents проигрывателя присвоено значение true ("1")
MouseMove	Возникает при перемещении курсора мыши над проигрывателем. Обработчику передаются следующие параметры: обозначение нажатой кнопки мыши (если она была нажата) — сумма чисел 1 (левая кнопка), 2 (правая кнопка) и 4 (средняя кнопка); обозначение нажатой клавиши-модификатора — сумма чисел 1 (была нажата клавиша <Shift>), 2 (клавиша <Ctrl>) и 4 (клавиша <Alt); горизонтальная координата точки, в которой находился курсор мыши, и ее вертикальная координата (обе координаты отсчитываются относительно "окна" проигрывателя). Все параметры передаются в числовом виде. Это событие генерируется только в том случае, если свойству (параметру) SendMouseMoveEvents проигрывателя присвоено значение true ("1")

Таблица 11.8 *(продолжение)*

Событие	Описание
MouseUp	Возникает при отпускании нажатой ранее кнопки мыши, если ее курсор находится над проигрывателем. Обработчику передаются следующие параметры: обозначение отпущенной кнопки мыши — сумма чисел 1 (левая кнопка), 2 (правая кнопка) и 4 (средняя кнопка); обозначение нажатой клавиши-модификатора — сумма чисел 1 (была нажата клавиша <Shift>), 2 (клавиша <Ctrl>) и 4 (клавиша <Alt); горизонтальная координата точки, в которой находился курсор мыши, и ее вертикальная координата (обе координаты отсчитываются относительно "окна" проигрывателя). Все параметры передаются в числовом виде. Это событие генерируется только в том случае, если свойству (параметру) SendMouseClickEvents проигрывателя присвоено значение true ("1")
NewStream	Возникает при начале загрузки нового мультимедийного файла
OpenStateChange	Возникает при изменении состояния открытия мультимедийного файла. Обработчику передаются два параметра — предыдущее и новое состояния. Эти параметры могут принимать значения: 0 (файл не открыт), 1 (идет загрузка файла описания потокового аудио или видео), 2 (идет загрузка файла описания интернет-радиостанции), 3 (поиск сервера), 4 (соединение с сервером), 5 (открытие файла) и 6 (файл открыт). Это событие генерируется только в том случае, если свойству (параметру) SendOpenStateChangeEvents проигрывателя присвоено значение true ("1")
PlayStateChange	Возникает при изменении состояния воспроизведения мультимедийного файла. Обработчику передаются два параметра — предыдущее и новое состояния. Эти параметры могут принимать значения: 0 (воспроизведение остановлено), 1 (воспроизведение приостановлено), 2 (файл воспроизводится), 3 (идет буферизация потока), 4 (идет прокрутка вперед), 5 (идет прокрутка назад), 6 (выполняется переход на следующую позицию), 7 (выполняется переход на предыдущую позицию) и 8 (файл не был открыт). Это событие генерируется только в том случае, если свойству (параметру) SendPlayStateChangeEvents проигрывателя присвоено значение true ("1")
PositionChange	Возникает при изменении посетителем позиции бегунка. Обработчику передаются два параметра — предыдущее и новое состояния в числовом виде
ReadyStateChange	Возникает при изменении состояния готовности проигрывателя к воспроизведению мультимедийного файла. Обработчику передается единственный параметр — новое состояние. Он может принимать значения: 0 (файл не был задан, т. е. свойству FileName не был присвоен его интернет-адрес), 1 (идет загрузка файла), 3 (загружен не весь файл, но воспроизведение можно начинать), 4 (файл загружен полностью)

Таблица 11.8 *(окончание)*

Событие	Описание
`Warning`	Возникает, когда проигрыватель сталкивается с возможной проблемой. Обработчику передаются параметры: тип проблемы (0 — звуковое устройство недоступно, 1 — неизвестный формат файла, 2 — файл не может быть воспроизведен), дополнительная информация о проблеме в виде числа и строковое описание проблемы. Это событие генерируется только в том случае, если свойству (параметру) `SendWarningEvents` проигрывателя присвоено значение `true` ("1")

Однако не все так просто с событиями проигрывателя Windows Media... Для привязки к ним обработчиков нужно использовать не хорошо знакомые нам по *главе 2* приемы, а особый синтаксис, поддерживаемый только Internet Explorer. Сейчас мы его рассмотрим.

Вот формат записи обработчиков событий в стиле Internet Explorer:

```
<SCRIPT FOR="<Имя проигрывателя>"
➥EVENT="<Событие и передаваемые им параметры>">
  <Тело обработчика>
</SCRIPT>
```

Ну, с *Именем проигрывателя* все ясно. А вот насчет *События и передаваемых им параметров* нужны дополнительные пояснения. Записаны они должны быть как объявление функции JavaScript, а именно, так:

```
<Имя события>([<Список передаваемых событием параметров, разделенных
➥запятыми>])
```

Впоследствии записанные в скобках параметры мы можем использовать в *Теле обработчика*.

> ***Народ советует***
>
> Полный список событий проигрывателя Windows Media можно узнать на посвященной ему странице сайта MSDN (**http://msdn.microsoft.com/library/en-us/wmp6sdk/htm/microsoftwindowsmediaplayercontrolversion64sdk.asp**).

А проигрыватель Shockwave/Flash, судя по всему, вообще не поддерживает событий. Автор обнаружил их список в разделе поддержки (**http://www.macromedia.com/support/flash**) Flash сайта фирмы Adobe, но так и не смог заставить их работать. Происходило это из-за новой модели безопасности, реализованной в версии 8 проигрывателя, или из-за чего-то еще, автору выяснить не удалось.

Пример

Далее приведен HTML-код еще одной Web-страницы, выводящей на экран продолжительность воспроизводимого в проигрывателе Windows Media аудиоклипа.

```
<HTML>
  <HEAD>
    <TITLE>Музыка на Web-странице</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
    ↳ID="acx">
      <PARAM NAME="FileName" VALUE="sound.wav">
    </OBJECT>
    <P ID="output">Продолжительность аудиоклипа:</P>
    <SCRIPT FOR="acx" EVENT="ReadyStateChange(pState)">
      var acxObject = document.all["acx"];
      var outputObject = document.all["output"];
      if (pState == 4)
        outputObject.firstChild.nodeValue += " " +
        ↳acxObject.Duration.toString() + " секунд.";
    </SCRIPT>
  </BODY>
</HTML>
```

Обратим внимание на написание обработчика события `ReadyStateChange` — это и есть обработчик событий в стиле Internet Explorer.

Как выяснить, установлен ли нужный модуль расширения?

Проблема

Я собираюсь создать Web-страницу с мультимедийным элементом и хочу написать сценарий, который будет выяснять, установлен ли на компьютере клиента нужный модуль расширения Web-обозревателя. Это можно сделать?

Решение (Opera и Firefox)

Пользователям Opera и Firefox в этом случае повезло — оба этих Web-обозревателя поддерживают коллекцию `mimeTypes`, вложенную в объект `navigator`. Эта коллекция содержит все типы данных, которые поддерживает Web-обозреватель, сам или с помощью модулей расширения.

Нужный тип данных задается в формате *MIME* (Multipurpose Internet Mail Extensions, многоцелевые расширения почты Интернета). Некоторые из типов MIME, соответствующие наиболее распространенным типам мультимедийных данных, приведены в табл. 11.9.

Таблица 11.9. *Типы MIME и соответствующие им форматы данных*

Тип файлов	Тип MIME
Аудио- или видеозапись ASF	video/x-ms-asf
Аудио- или видеозапись WMV	video/x-ms-wmv
Аудиозапись AIFF	audio/aiff
Аудиозапись AU	audio/basic
Аудиозапись MIDI	audio/mid
Аудиозапись MP3	audio/mpeg
Аудиозапись WAV	audio/wav
Аудиозапись WMA	audio/x-ms-wma
Видеозапись AVI	video/avi
Видеозапись Indeo (IVF)	video/x-ivf
Видеозапись MPEG	video/mpeg
Графический файл ART	image/x-jg
Графический файл BMP	image/bmp
Графический файл GIF	image/gif
Графический файл JPEG	image/jpeg
Графический файл Shockwave/Flash	application/x-shockwave-flash
Графический файл TIFF	image/tiff

Пример

Далее приведен HTML-код Web-страницы с видеоклипом. При открытии этой страницы выполняется проверка, установлен ли на компьютере подходящий для данного типа модуль расширения Web-обозревателя, и, если нет, выдается окно-сообщение.

```
<HTML>
  <HEAD>
    <TITLE>Пример</TITLE>
  </HEAD>
```

```
  <BODY>
    <SCRIPT TYPE="text/javascript">
      if (navigator.mimeTypes["video/avi"])
        document.write("<EMBED SRC=\"clip.avi\"></EMBED>")
      else
        window.alert("Нужный модуль расширения не установлен.");
    </SCRIPT>
  </BODY>
</HTML>
```

Как использовать фильтры и преобразования Internet Explorer?

Проблема

Я что-то слышал о фильтрах и преобразованиях, поддерживаемых Internet Explorer и позволяющих создавать на страницах весьма впечатляющие статичные и анимированные эффекты. Как я могу их использовать?

Решение

Исключительно просто! Но только нужно иметь в виду, что фильтры и преобразования поддерживаются только Internet Explorer 5.5 и более новыми версиями этой программы.

Итак, фильтры и преобразования. Что это такое и чем они отличаются? *Фильтр* — это особый эффект, применяемый к элементу страницы, причем эффект статичный. Это может быть тень, размытие, негативное отображение или что-то еще. Фильтры создавать проще всего — для этого не требуется никакого программирования.

В отличие от фильтра, *преобразование* — это динамичный эффект, скажем, плавное "перетекание" одного изображения или абзаца текста в другой. Создаются они примерно так же, как и фильтры, но требуют небольшого программирования (но именно небольшого).

Чтобы задать для какого-либо элемента страницы фильтр или преобразование, сначала нужно задать для этого элемента ширину и высоту. Это следует выполнить с помощью давно знакомых нам атрибутов стиля `width` и `height`. Это обязательно — иначе Internet Explorer не сможет применить к элементу страницы фильтр или преобразование.

Задав ширину и высоту элемента страницы, можно приступить к созданию самого фильтра или преобразования. Для этого достаточно вписать в определение стиля этого элемента особый атрибут `filter`.

Формат его записи таков:

```
filter: progid:DXImageTransform.Microsoft.
↳<Имя фильтра или преобразования>
↳([<Список свойств фильтра или преобразования>])
```

Список свойств фильтра и преобразования должен быть записан в виде пар <*Свойство*>=<*Значение*>, разделенных запятыми.

Далее приведен фрагмент HTML-кода, помещающего на страницу графическое изображение и применяющего к нему фильтр "размытие" (`MotionBlur`):

```
<IMG SRC="image.gif" ID="img" STYLE="height: 100; width: 100; filter:
↳progid:DXImageTransform.Microsoft.MotionBlur(Strength=50)">
```

Так, с фильтрами все более-менее ясно. А как же преобразования? А преобразования создаются точно так же, как фильтры, — с помощью атрибута стиля `filter`. Только для того, чтобы они нормально работали, нам придется чуть-чуть попрограммировать.

Прежде всего, нам понадобится коллекция `filters`, содержащая все примененные к элементу страницы фильтры. Эта коллекция вложена в объекты, представляющие все элементы страницы, поддерживающие фильтры (а это практически все видимые элементы страницы).

Чтобы создать работающее преобразование, нам нужно выполнить последовательность операций, приведенную далее.

1. Вызвать не принимающий параметров и не возвращающий значения метод `Apply` нужного нам преобразования — элемента коллекции `filters`, чтобы зафиксировать текущее состояние элемента страницы.
2. Выполнить необходимые действия с элементом управления. Например, мы можем сделать его невидимым, присвоив свойству `visibility` объекта `style` значение `false` (после этого, когда мы запустим преобразование, элемент страницы плавно исчезнет с экрана). Также мы можем присвоить свойству `src` графического изображения интернет-адрес другого графического файла (тогда одно изображение плавно сменится другим).
3. Вызвать также не принимающий параметров и не возвращающий значения метод `Play` нужного нам преобразования — элемента коллекции `filters`, чтобы запустить это преобразование.

Далее приведен фрагмент кода страницы, создающий графическое изображение, которое плавно, в течение 2 секунд (значение свойства `Duration` преобразования `Fade`), исчезнет при щелчке на нем.

```
<IMG SRC="image.gif" ID="img" STYLE="height: 100; width: 100; filter:
↳progid:DXImageTransform.Microsoft.Fade(Duration=2)"
↳ONCLICK="startTrans();">
```

```
<SCRIPT TYPE="text/javascript">
  function startTrans()
    {
      var imgObject = document.all["img"];
      imgObject.filters[0].Apply();
      imgObject.style.visibility = "hidden";
      imgObject.filters[0].Play();
    }
</SCRIPT>
```

Кроме того, все преобразования поддерживают не принимающий параметров и не возвращающий значения метод `Stop`, позволяющий остановить выполнение этого преобразования.

Но настала, наконец, пора ближе познакомиться как с самими фильтрами и преобразованиями, так и с поддерживаемыми ими свойствами и методами. Далее приведены две большие таблицы, где все они перечислены и описаны. В табл. 11.10 перечислены все фильтры, а в табл. 11.11 — преобразования.

Таблица 11.10. *Фильтры, поддерживаемые Internet Explorer*

Фильтр	Описание
`Alpha`	Делает элемент страницы прозрачным
`AlphaImageLoader`	"Подкладывает" графическое изображение под содержимое элемента страницы
`BasicImage`	Делает элемент страницы черно-белым, как бы просвеченным рентгеновскими лучами, и поворачивает его. Отдельно можно задавать угол поворота, степень "просвеченности" и т. п.
`Blur`	Делает элемент страницы размытым
`Chroma`	Делает заданный цвет элемента страницы прозрачным
`Compositor`	Объединяет цвета двух элементов страницы и выводит результат объединения
`DropShadow`	Заставляет элемент страницы "отбросить" тень заданного цвета, причем тень отображается отдельно от самого элемента
`Emboss`	Отображает элемент страницы выпуклым
`Engrave`	Отображает элемент страницы вдавленным
`Glow`	Отображает элемент страницы так, что он кажется тлеющим
`Gradient`	Градиентно закрашивает элемент страницы
`Light`	Отображает элемент страницы так, что он кажется освещенным
`MaskFilter`	Отображает прозрачный цвет элемента страницы цветом, заданным свойством `Color`, а все непрозрачные цвета делает прозрачными

Таблица 11.10 *(окончание)*

Фильтр	Описание
`Matrix`	Изменяет размеры элемента страницы, поворачивает или инвертирует его, используя матричные преобразования
`MotionBlur`	Делает элемент страницы размытым, словно быстро движущимся в заданном направлении
`Pixelate`	Разбивает элемент страницы на отдельные пикселы
`Shadow`	Заставляет элемент страницы "отбросить" тень в заданном направлении и заданного цвета
`Wave`	Создает волнистое искажение элемента страницы

Таблица 11.11. *Преобразования, поддерживаемые Internet Explorer*

Преобразование	Описание
`Barn`	Создает эффект "открывающейся" и "закрывающейся" "двери"
`BlendTrans`	Плавно заменяет старое содержимое элемента страницы на новое
`Blinds`	Создает эффект "открывающихся" и "закрывающихся" в заданном направлении "жалюзи"
`CheckerBoard`	Создает эффект "шахматной доски", движущейся в заданном направлении
`Fade`	Создает эффект *наплыва* (старое содержимое элемента страницы плавно пропадает, а новое одновременно так же плавно появляется)
`GradientWipe`	Новое содержимое элемента страницы "наползает" на старое, причем граница между ними выглядит как градиентная цветная полоса
`Inset`	Новое содержимое элемента страницы диагонально "наползает" на старое
`Iris`	Создает эффект ирисовой диафрагмы
`Pixelate`	Старое содержимое элемента страницы "рассыпается" на отдельные пикселы и пропадает, а новое "собирается" из отдельных пикселов. Аналогично фильтру `Pixelate`, описанному в табл. 11.10, оно анимировано
`RadialWipe`	Новое содержимое элемента страницы радиально "наползает" на старое
`RandomBars`	Старое содержимое элемента страницы "рассыпается" на отдельные линии и пропадает, а новое "собирается" из отдельных линий
`RandomDissolve`	Новое содержимое элемента страницы попиксельно "проявляется" на месте старого

Таблица 11.11 *(окончание)*

Преобразование	Описание
RevealTrans	Плавно заменяет старое содержимое элемента управления на новое, используя заданный эффект
Slide	Старое содержимое элемента страницы "сдвигается" в сторону, открывая под собой новое
Spiral	Новое содержимое элемента страницы спирально закрашивает старое
Stretch	Новое содержимое элемента страницы "растягивается", заменяя собой старое
Strips	Новое содержимое элемента страницы диагонально отдельными полосками "наползает" на старое
Wheel	Новое содержимое элемента страницы посекторно "наползает" на старое
Zigzag	Новое содержимое элемента страницы зигзагообразно, отдельными полосками "наползает" на старое

В табл. 11.12 приведены свойства, а в табл. 11.15 — методы, поддерживаемые фильтрами и преобразованиями.

Таблица 11.12. *Свойства фильтров и преобразований*

Свойство	Фильтр или преобразование	Описание
Add	MotionBlur, Wave	Если true, исходное содержимое элемента страницы перекрывает новое, если false (значение по умолчанию) — наоборот
Bands	Blinds, Slide	Количество полосок
Bias	Emboss, Engrave	Процентное значение, добавляемое к цвету элемента для получения цвета затенения
Color	Chroma, DropShadow, Glow, MaskFilter, Shadow	Цвет
Direction	MotionBlur, Shadow	Направление. Задается в градусах и должно быть кратно 45
Direction	Blinds, CheckerBoard	Направление. Может быть одним из четырех строковых значений: "up" (вверх), "down" (вниз), "right" (вправо) и "left" (влево)
Duration	Все преобразования	Продолжительность преобразования в секундах

Таблица 11.12 *(продолжение)*

Свойство	Фильтр или преобразование	Описание
Dx, Dy	Matrix	Значения fDx и fDy матричных преобразований
Enabled	Все фильтры и преобразования	Если true (значение по умолчанию), то фильтр или преобразование применяется к элементу страницы, если false — не применяется
EndColor, EndColorStr	Gradient	Конечный цвет градиентной закраски
FilterType	Matrix	Задает тип пикселов нового содержимого: "bilinear" (значение по умолчанию) или "nearest neighbor"
FinishOpacity	Alpha	Конечный уровень градиентной прозрачности; может быть от 0 (полная прозрачность; значение по умолчанию) до 100 (полная непрозрачность)
FinishX, FinishY	Alpha	Горизонтальная и вертикальная координаты позиции, в которой заканчивается область градиентной прозрачности
Freq	Wave	Количество "волн"
Function	Compositor	Функция преобразования. Список всех возможных значений приведен в табл. 11.13
GradientSize	GradientWipe	Часть площади элемента страницы, покрываемой градиентной полосой. Может быть от 0.0 до 1.0
GradientType	Gradient	Если 1 (значение по умолчанию), градиентная заливка располагается по горизонтали, если 0 — по вертикали
GrayScale	BasicImage	Если 1, элемент страницы отображается черно-белым, если 0 (значение по умолчанию) — цветным
GridSizeX, GridSizeY	Spiral, Zigzag	Количество полосок по горизонтали или вертикали. Может быть от 1 до 100
Invert	BasicImage	Если 1, элемент страницы отображается с инвертированными цветами, если 0 (значение по умолчанию) — как обычно
IrisStyle	Iris	Форма апертуры "ирисовой диафрагмы". Доступны значения: "DIAMOND" (бриллиант), "CIRCLE" (круг), "CROSS" (X-образная), "PLUS" (знак "плюс"; значение по умолчанию), "SQUARE" (квадратная) и "STAR" (звездообразная)

Таблица 11.12 *(продолжение)*

Свойство	Фильтр или преобразование	Описание
`LightStrength`	`Wave`	Постоянство окраски "волн". Может быть от 0 до 100
`M11`, `M12`, `M21`, `M22`	`Matrix`	Значения `fM11`, `fM12`, `fM21` и `fM22` матричных преобразований
`MakeShadow`	`Blur`	Если `true`, элемент страницы будет затенен, если `false` (значение по умолчанию) — не будет
`Mask`	`BasicImage`	Если 1, прозрачный цвет элемента страницы будет заменен значением свойства `MaskColor`, если 0 (значение по умолчанию) — не будет
`MaskColor`	`BasicImage`	Цвет, на который будет заменен прозрачный цвет элемента страницы
`MaxSquare`	`Pixelate`	Максимальный размер пиксела
`Mirror`	`BasicImage`	Если 1, элемент страницы будет отображен зеркально, если 0 (значение по умолчанию) — как обычно
`Motion`	`Barn`, `Iris`	Если `"out"` (значение по умолчанию), движение будет происходить от центра элемента страницы к его границам, если `"in"` — от границ к центру
`Motion`	`Strip`	Угол наклона полосок и направление движения нового содержимого. Доступны значения: `"leftdown"` (полоски наклонены влево, движение выполняется вниз), `"leftup"` (влево и вверх), `"rightdown"` (вправо и вниз) и `"rightup"` (вправо и вверх)
`Motion`	`GradientWipe`	Если `"forward"` (значение по умолчанию), движение производится согласно значению свойства `WipeStyle`, если `"reverse"` — в обратном направлении
`OffX`, `OffY`	`DropShadow`	Горизонтальное и вертикальное смещения тени
`Opacity`	`Alpha`	Начальный уровень градиентной прозрачности; может быть от 0 (полная прозрачность; значение по умолчанию) до 100 (полная непрозрачность)
`Opacity`	`BasicImage`	Уровень прозрачности элемента страницы. Может быть от 0.0 до 1.0
`Orientation`	`Barn`, `RandomBars`	Если `"horizontal"`, преобразование происходит по горизонтали, если `"vertical"` — по вертикали

Таблица 11.12 *(продолжение)*

Свойство	Фильтр или преобразование	Описание
`Overlap`	`Fade`	Время, относительно общей продолжительности преобразования, когда и старое, и новое содержимое элемента страницы отображаются одновременно. Может быть от 0.0. до 1.0
`Percent`	Все преобразования	Процент выполнения преобразования. Может быть от 0 (преобразование еще не начиналось) до 100 (преобразование закончено)
`Phase`	`Wave`	Фаза "волн". Может быть от 0 до 100
`PixelRadius`	`Blur`	Размер области "размытия". Может быть от 1.0 до 100.0
`Positive`	`DropShadow`	Если `true` (значение по умолчанию), тень создается из прозрачных пикселов элемента страницы, если `false` — из непрозрачных ("негативная" тень)
`Rotation`	`BasicImage`	Угол поворота элемента страницы. Доступны значения: 0 (значение по умолчанию) — нет поворота, 1 — на 90º, 2 — на 180º и 3 — на 270º
`ShadowOpacity`	`Blur`	Прозрачность тени. Может быть от 0.0 (полностью прозрачная) до 1.0 (полностью непрозрачная)
`SizingMethod`	`AlphaImageLoader`	Способ размещения изображения в границах элемента страницы. Доступны значения: `"crop"` (обрезание изображения), `"image"` (уменьшение или увеличение самого элемента страницы; значение по умолчанию) и `"scale"` (уменьшение или увеличение изображения)
`SizingMethod`	`Matrix`	Способ размещения нового изображения в границах элемента страницы. Доступны значения: `"clip to original"` (обрезание изображения; значение по умолчанию) и `"auto expand"` (уменьшение или увеличение изображения)
`SlideStyle`	`Slide`	Способ замещения содержимого элемента страницы. Доступны значения: `"HIDE"` (скрытие; значение по умолчанию), `"PUSH"` ("выталкивание") и `"SWAP"` (замена)
`Spokes`	`Wheel`	Количество секторов. Может быть от 2 до 20

Таблица 11.12 *(окончание)*

Свойство	Фильтр или преобразование	Описание
SquaresX, SquaresY	CheckerBoard	Количество рядов по горизонтали и вертикали
src	AlphaImageLoader	Интернет-адрес файла изображения
StartColor, StartColorStr	Gradient	Начальный цвет градиентной закраски
StartX, StartY	Alpha	Горизонтальная и вертикальная координаты позиции, в которой начинается область градиентной прозрачности
status	Все преобразования	Возвращает состояние выполнения преобразования. Может принимать значения: 0 (преобразование было остановлено вызовом описанного в табл. 11.5 метода Stop), 1 (преобразование завершено) и 2 (преобразование еще выполняется)
Strength	MotionBlur, Glow, Wave	Дистанция в пикселах, на которой выполняется преобразование
StretchStyle	Stretch	Способ замещения содержимого элемента страницы. Доступны значения: "HIDE" (скрытие), "PUSH" ("выталкивание") и "SPIN" (замена; значение по умолчанию)
Style	Alpha	Форма градиентной прозрачности. Доступны значения: 0 (нет градиента; значение по умолчанию), 1 (линейный градиент), 2 (круговой градиент) и 3 (прямоугольный градиент)
Transition	RevealTrans	Эффект преобразования. Список всех возможных значений приведен в табл. 11.14
WipeStyle	RadialWipe	Способ замещения содержимого элемента страницы. Доступны значения: "CLOCK" (вращение вокруг центра элемента страницы по часовой стрелке; значение по умолчанию), "WEDGE" (вращение сразу в обе стороны) и "RADIAL" (радиальное вращение)
WipeStyle	GradientWipe	Если 0 (значение по умолчанию), движение происходит по горизонтали, если 1 — по вертикали
XRay	BasicImage	Если 1, то элемент страницы будет отображен "просвеченным" "рентгеновскими лучами", если 0 (значение по умолчанию) — как обычно

Таблица 11.13. *Доступные значения свойства* `Function` *преобразований*

Значение	Описание
0	Никакая операция не выполняется. Значение по умолчанию
1	Сравнивает накладывающиеся пикселы обоих изображений и выводит пиксел с меньшей яркостью
2	Сравнивает накладывающиеся пикселы обоих изображений и выводит пиксел с большей яркостью
3	Выводит только первое изображение
4	Выводит первое изображение поверх второго. Второе изображение будет видно сквозь прозрачные участки первого
5	Выводит те пикселы первого изображения, которым соответствуют непрозрачные пикселы второго
6	Выводит те пикселы первого изображения, которым соответствуют прозрачные пикселы второго
7	Выводит первое изображение, чьи пикселы имеют такую же прозрачность, что и соответствующие пикселы второго изображения
8	Сначала вычитает цвет каждого пиксела второго изображения из цвета соответствующего пиксела первого. После этого задает для каждого пиксела результирующего изображения прозрачность соответствующих пикселов второго изображения. Выводит результирующее изображение
9	Сначала складывает цвет каждого пиксела первого изображения с цветом соответствующего пиксела второго. После этого задает для каждого пиксела результирующего изображения прозрачность соответствующих пикселов второго изображения. Выводит результирующее изображение
10	Выполняет операцию "исключающее ИЛИ" (XOR) с пикселами обоих изображений и выводит результат
19	Выводит только второе изображение
20	Выводит второе изображение поверх первого. Первое изображение будет видно сквозь прозрачные участки второго
21	Выводит те пикселы второго изображения, которым соответствуют непрозрачные пикселы первого
22	Выводит те пикселы второго изображения, которым соответствуют прозрачные пикселы первого
23	Выводит второе изображение, чьи пикселы имеют такую же прозрачность, что и соответствующие пикселы первого изображения
24	Сначала вычитает цвет каждого пиксела первого изображения из цвета соответствующего пиксела второго. После этого задает для каждого пиксела результирующего изображения прозрачность соответствующих пикселов первого изображения. Выводит результирующее изображение

Таблица 11.13 *(окончание)*

Значение	Описание
25	Сначала складывает цвет каждого пиксела второго изображения с цветом соответствующего пиксела первого. После этого задает для каждого пиксела результирующего изображения прозрачность соответствующих пикселов первого изображения. Выводит результирующее изображение

Таблица 11.14. *Доступные значения свойства* `Transition` *преобразований*

Значение	Описание
0	Увеличивающийся прямоугольник
1	Уменьшающийся прямоугольник
2	Увеличивающийся круг
3	Уменьшающийся круг
4	Подъем снизу
5	Соскальзывание сверху
6	Сдвиг справа
7	Сдвиг слева
8	Вертикальные "жалюзи"
9	Горизонтальные "жалюзи"
10	"Шахматная доска" с увеличивающимися клетками
11	"Шахматная доска" с уменьшающимися клетками
12	Размытие и появление
13	"Раскрывающаяся дверь" по вертикали
14	"Закрывающаяся дверь" по вертикали
15	"Раскрывающаяся дверь" по горизонтали
16	"Закрывающаяся дверь" по горизонтали
17	"Наезд" из левого нижнего угла
18	"Наезд" из левого верхнего угла
19	"Наезд" из правого нижнего угла
20	"Наезд" из правого верхнего угла
21	Горизонтальные полосы
22	Вертикальные полосы
23	Случайный выбранный из приведенных выше эффект

Таблица 11.15. *Методы фильтров и преобразований*

Метод	Фильтр или преобразование	Описание
`AddAmbient(<Красный>, <Зеленый>, <Синий>, <Интенсивность>)`	`Light`	Добавляет источник рассеянного света с заданными цветовыми параметрами
`AddCone(<X1>, <Y1>, <Z1>, <X2>, <Y2>, <Красный>, <Зеленый>, <Синий>, <Интенсивность>, <Угол>)`	`Light`	Добавляет источник направленного света с заданными цветовыми параметрами. *X1*, *Y1* и *Z1* задают координаты источника, а *X2* и *Y2* — точки, куда падает свет
`AddPoint(<X>, <Y>, <Z>, <Красный>, <Зеленый>, <Синий>, <Интенсивность>)`	`Light`	Добавляет источник направленного света с заданными цветовыми параметрами. *X*, *Y* и *Z* задают координаты источника
`Apply()`	Все преобразования	Фиксирует текущее состояние элемента страницы, после чего мы можем делать с ним все, что хотим. Для запуска преобразования следует использовать описанный далее метод `Play`
`ChangeColor(<Номер>, <Красный>, <Зеленый>, <Синий>, <Тип значения>)`	`Light`	Изменяет цвет источника света с заданным *Номером*. Последний параметр задает абсолютное (1 или любое ненулевое значение) или относительное (0) изменение цвета
`ChangeStrength(<Номер>, <Интенсивность>, <Тип значения>)`	`Light`	Изменяет интенсивность источника света с заданным *Номером*. Последний параметр задает абсолютное (1 или любое ненулевое значение) или относительное (0) изменение интенсивности
`Clear()`	`Light`	Удаляет все источники света
`MoveLight(<Номер>, <X>, <Y>, <Z>, <Тип значения>)`	`Light`	Перемещает источник света с заданным *Номером*. Последний параметр задает абсолютное (1 или любое ненулевое значение) или относительное (0) перемещение
`Play(<Продолжительность>)`	Все преобразования	Запускает преобразование. Единственный параметр задает продолжительность преобразования в секундах

Таблица 11.15 *(окончание)*

Метод	Фильтр или преобразование	Описание
`Stop()`	Все преобразования	Останавливает преобразование

Также все элементы страницы поддерживают событие `onFilterChange`, возникающее при изменении состояния или завершении работы фильтра или преобразования. Мы можем использовать это событие, например, для того, чтобы присвоить элементу страницы другой фильтр или другое преобразование.

Народ предупреждает!

Не забываем, что фильтры и преобразования поддерживаются только Internet Explorer 5.5 и более новыми версиями этой программы.

Народ замечает

Вообще-то, фильтры и преобразования начали поддерживаться еще Internet Explorer 4.0. Но, во-первых, там использовался другой формат записи, не совместимый с современным, а во-вторых, доля Internet Explorer 4.0 на рынке сейчас очень невелика, так что вряд ли стоит принимать в расчет эту старую программу.

Народ советует

Желающие подробнее узнать о фильтрах и преобразованиях могут наведаться в посвященный им раздел сайта MSDN (**http://msdn.microsoft.com/workshop/author/filter/filters_transitions_entry.asp**).

Пример 1

Далее приведен HTML-код Web-страницы с графическим изображением, к которому применен эффект размытия (`Blur`).

```
<HTML>
  <HEAD>
    <TITLE>Размытый рисунок</TITLE>
  </HEAD>
  <BODY>
    <IMG SRC="image.gif" STYLE="height: 100; width: 100; filter:
    progid:DXImageTransform.Microsoft.Blur(PixelRadius=3)">
  </BODY>
</HTML>
```

Вот так, всего лишь одной строкой CSS-кода, мы создали на Web-странице красивый графический эффект. Жаль, что его сможет отобразить только Internet Explorer 5.5 или более новых версий...

Пример 2

А теперь давайте рассмотрим Web-страницу с заголовком, отбрасывающим тень. Этот эффект создается с помощью фильтра `Shadow`.

```
<HTML>
  <HEAD>
    <TITLE>Бросим тень на Web-страницу!</TITLE>
    <STYLE>
      H1 { filter: progid:DXImageTransform.Microsoft.Shadow
           ↳(Color=#CCCCCC, Direction=135);
           height: 50px;
           width: 400px; }
    </STYLE>
  </HEAD>
  <BODY>
    <H1>Это заголовок с тенью</H1>
  </BODY>
</HTML>
```

Здесь мы поместили определение фильтра прямо в стиль переопределения тега `<H1>`. Это значит, что все заголовки первого уровня на этой странице будут отбрасывать тень. И опять нам для этого понадобилось всего ничего — единственная строка CSS-кода. (Ну и еще две строки, задающие ширину и высоту заголовка — без этого, увы, нельзя...)

Пример 3

Последним примером будет Web-страница, выводящая последовательно, одно за другим, три графических изображения, которые будут плавно "перетекать" друг в друга. Это реализуется с помощью преобразования "наплыв" (`Fade`).

```
<HTML>
  <HEAD>
    <TITLE>Преобразование</TITLE>
    <SCRIPT TYPE="text/javascript">
      var imagesArray = new Array();
      imagesArray[0] = "image1.gif";
      imagesArray[1] = "image2.gif";
      imagesArray[2] = "image3.gif";
      var imageNumber = 0;
    </SCRIPT>
  </HEAD>
  <BODY>
    <IMG SRC="" ID="img" STYLE="height: 100; width: 100; filter:
    ↳progid:DXImageTransform.Microsoft.Fade(Duration=1)">
```

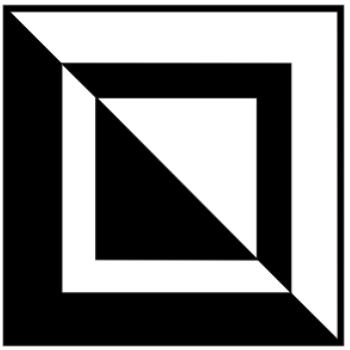
```
    <SCRIPT TYPE="text/javascript">
      var imgObject = document.all["img"];
      imgObject.src = imagesArray[imageNumber];
      imgOnFilterChange();

      function imgChange()
        {
          if (imageNumber >= imagesArray.length - 1)
            imageNumber = 0
          else
            imageNumber++;
          imgObject.filters[0].Apply();
          imgObject.src = imagesArray[imageNumber];
          imgObject.filters[0].Play();
        }

      function imgOnFilterChange()
        {
          window.setTimeout(imgChange, 1000);
        }

      imgObject.onfilterchange = imgOnFilterChange;
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь мы просто каждую секунду меняем изображение, отображаемое в теге `<IMG>`, и применяем к этому тегу преобразование `Fade`. Для отслеживания окончания преобразования мы используем событие `onFilterChange`. Кроме того, чтобы изображения менялись не очень часто (иначе у посетителя зарябит в глазах), мы задаем секундную задержку перед их сменой (для этого мы использовали хорошо нам знакомый метод `setTimeout` объекта `window`). В общем, ничего сложного — зато какой красивый эффект!

Что дальше?

Ну вот, собственно, и все о работе с Web-страницами и их элементами... Работали мы с ними весьма много, и результаты этой работы оказались впечатляющими. Берегитесь, Web-страницы и Web-обозреватели — идут Web-программисты!

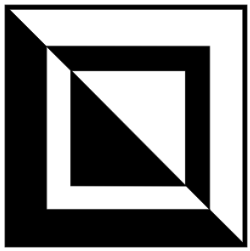
В следующей части этой книги мы займемся работой с данными. В смысле, с теми данными, что нужно вывести посетителю или принять от него. Разговор о них будет хоть и не таким долгим, как об элементах Web-страниц, но тоже плодотворным. Начнем же!



ЧАСТЬ IV

Работа с данными

ГЛАВА 12



Простейший ввод/вывод данных

Ну что ж, настала пора пообщаться с посетителем нашего Web-творения — принять от него данные или, скажем, вывести ему что-нибудь этакое. А то, пожалуй, заскучает посетитель и уйдет с нашего сайта...

Как всегда, народ — дока в подобных вопросах. Так что давайте посмотрим, как он справится с задачами ввода и вывода данных. Для начала — с самыми простыми.

Использование стандартных окон Web-обозревателя для ввода данных

Начнет народ со стандартных окон Web-обозревателя, позволяющих очень быстро и без особых трудов ввести и вывести какую-либо информацию. Конечно, это самые простые средства, но зачастую их будет достаточно, если выводимая информация не слишком сложна и объемна.

Как мне вывести строку текста или число?

Проблема

Я создал сценарий, выполняющий вычисления, и теперь не знаю, как показать вычисляемый им результат посетителю. Может быть, народ знает?

Решение

Использовать метод `alert` объекта `window`. Этот метод выводит на экран стандартное окно-сообщение Windows с заданным текстом и кнопкой **OK**.

Формат вызова этого метода очень прост:

```
window.alert(<Текст сообщения>);
```

Значения этот метод не возвращает.

Пример

```
window.alert("Приветствую вас на своем сайте!");
```

Это выражение выведет на экран окно-сообщение с текстом "Приветствую вас на своем сайте!".

Народ советует

Вообще, окна-сообщения следует использовать только в случае крайней нужды (например, чтобы сообщить посетителю, что введенные им в Web-форму данные некорректны). Просто так применять их не нужно. Не берите пример с автора этой книги.

Народ советует

Окна-сообщения, выводимые методом `alert` объекта `window`, неплохо подходят для отладки сценариев, когда нужно выяснить значение какой-либо переменной. Просто передаем эту переменную методу `alert` и в нужный момент получаем результат.

Также метод `alert` объекта `window` подойдет, когда нужно выяснить, выполняется ли какой-то фрагмент кода JavaScript. Вставляем вызов этого метода в какое-либо место данного фрагмента кода (в качестве значения ему можно передать любую строку, например, "!!!" или "Код выполнен") и смотрим, появилось ли в нужный момент окно-сообщение. Если не появилось, значит, мы где-то допустили ошибку.

Как предложить посетителю выбор из двух альтернатив?

Проблема

Мой сценарий должен вычислять разные значения или выполнять разные действия в зависимости от выбора посетителя. Так как мне выяснить, что хочет выбрать посетитель?

Решение

Воспользоваться методом `confirm` объекта `window`. Он выводит на экран стандартное окно-сообщение Windows с текстом и кнопками **OK** и **Отмена** (Cancel).

Формат вызова этого метода таков:

```
window.confirm(<Текст сообщения>);
```

Метод `confirm` возвращает `true`, если была нажата кнопка **OK**, и `false`, если была нажата кнопка **Отмена** (Cancel).

Пример

```
if (window.confirm("Вы хотите перейти на другую Web-страницу?"))
  window.location.href = "other_page.html";
```

Приведенный выше сценарий спрашивает посетителя, хочет ли он перейти на другую страницу, и, если посетитель нажмет кнопку **OK**, выполняет переход.

> ***Народ советует***
>
> Здесь, опять же, действует то же самое правило: выводимые методом `confirm` окна-сообщения следует использовать только в случае действительной в них нужды, но никак не просто так, чтобы показать свою квалификацию Web-программиста.

Как запросить у посетителя данные?

Проблема

Могу ли я запросить у посетителя какие-либо данные, не применяя для этого Web-форму?

Решение

Это можно, но только в том случае, если нужно запросить лишь одно значение (строку или число). Для этого следует использовать метод `prompt` объекта `window`, который выводит на экран стандартное окно ввода Web-обозревателя с текстом, полем ввода и кнопками **OK** и **Отмена** (Cancel).

Формат вызова этого метода таков:

```
window.prompt(<Текст сообщения>[, <Изначальное значение>]);
```

А вот здесь требуются пояснения. Итак, первый, обязательный, параметр задает текст, который выводится в окне ввода. Второй же, необязательный, параметр задает изначальное значение, которое будет присутствовать в поле ввода этого окна.

> ***Народ предупреждает!***
>
> Если второй, необязательный, параметр метода `prompt` объекта `window` пропущен, в поле ввода окна будет присутствовать текст "Undefined", что может обескуражить посетителя. Поэтому лучше всегда его задавать. Если же в поле ввода не должно ничего присутствовать изначально, вторым параметром следует передать пустую строку.

Если посетитель ввел в поле ввода окна какой-либо текст и нажал кнопку **OK**, метод `prompt` вернет этот текст в строковом виде. Если же посетитель

нажал кнопку **Отмена** (Cancel), этот метод вернет значение `null`, даже если в поле ввода окна было что-то введено.

Пример

```
var name = window.prompt("Введите, пожалуйста, свое имя", "");
if (name)
  document.write("<P>Здравствуйте, " + name + "!</P>");
```

Приведенный выше небольшой сценарий запрашивает у посетителя его имя и, если оно было введено и посетитель нажал кнопку **OK** окна ввода, выводит его на Web-страницу.

Обратим внимание, что вторым параметром методу `prompt` мы по совету всезнающего народа передали пустую строку. В результате в поле ввода выведенного этим методом окна изначально не будет ничего присутствовать.

Народ советует

Окна ввода, выводимые методом `prompt` объекта `window`, вообще не стоит использовать. Как правило, их возможности невелики (много ли данных можно ввести в одно поле ввода...), да и само их наличие говорит о непрофессионализме Web-дизайнера.

Как вывести произвольный текст в строку статуса?

Проблема

Строка статуса окна Web-обозревателя всегда при деле — показывает интернет-адреса гиперссылок, на которые указывает курсор мыши, сообщает об окончании загрузки страниц и т. п. А могу ли я вывести в нее произвольный текст?

Решение 1

Запросто! Для этого нужно воспользоваться свойством `status` объекта `window`, которое как раз и задает отображаемый в строке статуса текст.

Чтобы отобразить в строке статуса текст по умолчанию (как правило, это надпись "Готово", сообщающая об окончании загрузки страницы), нужно присвоить свойству `status` объекта `window` значение свойства `defaultStatus` этого же объекта. (Свойство `defaultStatus` как раз и задает текст, отображаемый в строке статуса по умолчанию.)

Если задание нового значения свойства `status` выполняется в обработчике события, привязанном прямо к тегу, т. е. через соответствующий событию атрибут, в значении этого атрибута следует также добавить выражение, возвращающее значение `true`.

Вот так:

```
<<Тег> <Атрибут, соответствующий событию>="<Другие выражения
↳обработчика>; return true;">
```

Это нужно для того, чтобы обработчик события работал правильно.

Внимание: в функции-обработчике это выражение (в смысле, `return true`) не оказывает такого эффекта!

> ***Народ советует***
>
> Не следует злоупотреблять выводом своего текста в строку статуса. А уж если что-то туда и выводить, так сначала подумать, так ли уж это нужно.

> ***Народ предупреждает!***
>
> Некоторые современные Web-обозреватели, в частности Opera и Firefox, могут блокировать Web-сценарии, выводящие в строку статуса произвольный текст. Кроме того, некоторые программы фильтрации рекламы и всплывающих окон, например WebWasher и AdMuncher, тоже умеют это делать. Так что следует иметь это в виду.
>
> Вдобавок, в Opera текст строки статуса, заданный свойством `status`, почему-то не отображается. Автору книги так и не удалось выяснить, почему это происходит.

Пример

Далее приведен HTML-код Web-страницы с гиперссылкой, при наведении на которую курсора мыши в строке статуса окна Web-обозревателя выводится примечание к этой гиперссылке.

```
<HTML>
  <HEAD>
    <TITLE>Гиперссылка с примечанием</TITLE>
    <SCRIPT TYPE="text/javascript">
      function hrfOnMouseOver()
        {
          window.status = "Щелкните здесь для перехода на другую
          ↳страницу";
        }

      function hrfOnMouseOut()
        {
          window.status = window.defaultStatus;
        }
    </SCRIPT>
  </HEAD>
```

```
  <BODY>
    <P><A HREF="somepage.html" ONMOUSEOVER="hrfOnMouseOver();
    ↳return true;" ONMOUSEOUT="hrfOnMouseOut(); return true;">
    ↳На другую страницу</A></P>
  </BODY>
</HTML>
```

Обратим внимание, как мы записали значение атрибута `ONMOUSEOVER`, соответствующего одноименному событию. А мы записали вот так:

```
hrfOnMouseOver(); return true;
```

т. е. после вызова функции-обработчика `hrfOnMouseOver` указали также выражение `return true`, возвращающее Web-обозревателю значение `true`. Это нужно, чтобы код, изменяющий текст в строке статуса, действительно там появился. (Собственно, об этом было сказано ранее.)

Точно так же мы поступили, когда записывали значение атрибута `ONMOUSEOUT`. Хотя, судя по результатам экспериментов автора, в случае этого события (`onMouseOut`) это необязательно — код нормально работает и без выражения `return true`.

Решение 2

Использовать универсальную функцию `jspsSetStatusBarText` (листинг 12.1), чей формат вызова таков:

```
jspsSetStatusBarText(<Текст>[, <Тайм-аут>]);
```

Первым параметром функции передается сам текст, который нужно вывести в строку статуса. Вторым, необязательным, параметром передается величина тайм-аута, в течение которого этот текст будет присутствовать в строке статуса; его величина задается в секундах. Значение тайм-аута, равное 0, приводит к тому, что заданный текст будет отображаться в строке статуса постоянно, т. е. до того момента, пока его не заменит текст, выведенный самим Web-обозревателем или заданный программистом в Web-сценарии. Если второй параметр пропущен, заданный текст будет присутствовать в строке статуса две секунды.

Листинг 12.1. Функция `jspsSetStatusBarText`, выводящая заданный текст в строку статуса окна Web-обозревателя в течение заданного тайм-аута

```
// Объявление самой функции jspsSetStatusBarText
function jspsSetStatusBarText(pText, pTimeout)
  {
    if (typeof(pTimeout) == "undefined") pTimeout = 2;
    window.status = pText;
```

```
    if (pTimeout != 0)
      window.setTimeout(jspsSSBTRemoveText, pTimeout * 1000)
  }

// Объявление служебной функции, удаляющей заданный текст из строки
// статуса и заменяющей его текстом по умолчанию
function jspsSSBTRemoveText()
  {
    window.status = window.defaultStatus;
  }
```

Хорошая идея!

Поместите объявление этой функции в файл сценариев setstatusbartext.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="setstatusbartext.js"></SCRIPT>
```

Пример

Вот пример Web-страницы, выводящей после окончания загрузки в строку статуса сообщение и "удерживающей" его там одну секунду.

```
<HTML>
  <HEAD>
    <TITLE>Страница, сообщающая об окончании загрузки</TITLE>
    <SCRIPT SRC="setstatusbartext.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsSetStatusBarText
    находится в файле сценариев setstatusbartext.js -->
  </HEAD>
  <BODY>
    <P>Некоторое время в строке статуса этого окна Web-обозревателя будет
    ↳отображаться сообщение об окончании загрузки страницы.</P>
    <SCRIPT TYPE="text/javascript">
      jspsSetStatusBarText("Страница загружена", 1);
    </SCRIPT>
  </BODY>
</HTML>
```

Как задать произвольный текст по умолчанию для строки статуса?

Проблема

А как я могу задать свой текст, который выводился бы в строке статуса окна Web-обозревателя по умолчанию?

Решение

Очень просто! Достаточно присвоить нужное значение свойству `defaultStatus` объекта `window`. Это свойство как раз и "отвечает" за текст строки статуса по умолчанию.

Пример

```
window.defaultStatus = "Это моя страница!";
```

Приведенное выражение задает текст по умолчанию "Это моя страница!" для строки статуса.

Как вывести предупреждение для посетителя прямо на Web-страницу?

Проблема

Я создал на Web-странице форму и хочу теперь предупреждать посетителя, если он ввел в нее некорректные данные, но не хочу пользоваться окнами-сообщениями. Есть ли у меня выход?

Решение

Выход есть. Даже не один.

- В случае Web-формы можно выделять цветом элементы управления, в которые введены некорректные данные.
- Можно создать на Web-странице необходимый текст предупреждения, скажем, в виде абзаца и сделать его скрытым (для этого достаточно присвоить свойству `visibility` объекта `style` значение `"hidden"`). Чтобы показать это предупреждение посетителю, нужно присвоить этому же свойству значение `"visible"`. Как говорится, дешево и сердито.
- Можно использовать свободно позиционируемый элемент, содержащий текст предупреждения. Изначально его также следует сделать скрытым, а в случае нужды показать в нужном месте страницы (в ее центре или возле нужного элемента управления Web-формы).
- Можно использовать описанный в *главе 7* объект `PopupTip` (см. листинг 7.1). Этот объект позволяет выводить всплывающие подсказки для элементов страницы. Хотя этот способ годится, скорее, для вывода просто подсказок (например, указания, что и в каком формате следует вводить в данном поле ввода).
- Можно использовать универсальную функцию `jspsShowPopupAlert` (листинг 12.2). Эта функция неплохо подходит для вывода всплывающих предупреждений.

Народ предупреждает!

Использовать для вывода действительно важных предупреждений строку статуса окна Web-обозревателя не стоит, т. к. посетитель обычно туда не смотрит.

Вот формат вызова функции `jspsShowPopupAlert`:

```
jspsShowPopupAlert(<Текст>[, <Тайм-аут>[, <Выравнивание>
↳[, <Стилевой класс>]]]);
```

Первым, и единственным обязательным, параметром в эту функцию передается текст предупреждения. Здесь все ясно.

Вторым, необязательным, параметром передается величина тайм-аута, в течение которого предупреждение должно присутствовать на странице; его величина задается в секундах. Если он пропущен, предупреждение будет присутствовать на странице две секунды.

Третьим, также необязательным, параметром задается выравнивание предупреждения, которое представляет собой свободно позиционируемый элемент, на Web-странице. Доступны следующие значения, для задания которых используются псевдоконстанты, объявленные в листинге 10.1 (функция `jspsAlignContainer`):

- `JSPS_AC_LEFTTOP` — по левому и верхнему краю родителя (т. е. поместить элемент в его левый верхний угол);
- `JSPS_AC_RIGHTTOP` — по правому и верхнему краю (в правый верхний угол);
- `JSPS_AC_RIGHTBOTTOM` — по правому и нижнему краю (в правый нижний угол);
- `JSPS_AC_LEFTBOTTOM` — по левому и нижнему краю (в левый нижний угол);
- `JSPS_AC_CENTER` — по центру.

Остальные псевдоконстанты лучше не использовать, иначе точное положение предупреждения на странице будет непредсказуемым.

Если третий параметр пропущен, предупреждение будет выровнено по центру страницы (как если бы мы задали значение `JSPS_AC_CENTER`).

Последний, и также необязательный, параметр задает имя стилевого класса, который будет привязан к свободно позиционируемому элементу, формирующему предупреждение. Если он не задан, функция `jspsShowPopupAlert` сама присвоит свободному элементу необходимые атрибуты стиля.

В стилевом классе, который мы привязываем к свободно позиционируемому элементу — предупреждению, мы можем использовать любые атрибуты и любые их значения. В частности, мы можем задать цвет текста и фона, шрифт, выравнивание, рамку, отступы и границы. Единственное: нам не следует задавать в нем метод позиционирования (атрибут стиля `position`) и ко-

ординаты (атрибуты стиля `left` и `top`), поскольку это сделает сама функция `jspsShowPopupAlert`.

Народ предупреждает!

В данный момент на странице может присутствовать только одно выведенное функцией `jspsShowPopupAlert` предупреждение.

Листинг 12.2. Функция `jspsShowPopupAlert`, выводящая на экран предупреждение с заданным текстом

```
// Объявляем служебную псевдоконстанту, задающую имя, которое примет
// свободный элемент — предупреждение. Это имя понадобится служебной
// функции jspsSPARemovePopupAlert, удаляющей предупреждение
var JSPS_SPA_POPUP_ALERT_NAME = "jspsSPAPopupAlert";

// Объявление самой функции jspsShowPopupAlert
function jspsShowPopupAlert(pText, pTimeout, pAlignment, pClassName)
  {
    if (!pTimeout) pTimeout = 2;
    if (typeof(pAlignment) == "undefined") pAlignment = JSPS_AC_CENTER;
    var popupElement = document.createElement("DIV");
    popupElement.id = JSPS_SPA_POPUP_ALERT_NAME;
    if (pClassName)
      popupElement.className = pClassName
    else
      {
        popupElement.style.backgroundColor = "#FFCCCC";
        popupElement.style.borderLeftStyle = "solid";
        popupElement.style.borderLeftColor = "#990000";
        popupElement.style.borderLeftWidth = "1";
        popupElement.style.borderTopStyle = "solid";
        popupElement.style.borderTopColor = "#990000";
        popupElement.style.borderTopWidth = "1";
        popupElement.style.borderRightStyle = "solid";
        popupElement.style.borderRightColor = "#990000";
        popupElement.style.borderRightWidth = "1";
        popupElement.style.borderBottomStyle = "solid";
        popupElement.style.borderBottomColor = "#990000";
        popupElement.style.borderBottomWidth = "1";
        popupElement.style.paddingLeft = "10";
        popupElement.style.paddingTop = "10";
        popupElement.style.paddingRight = "10";
        popupElement.style.paddingBottom = "10";
      }
```

```
    popupElement.style.position = "absolute";
    var alertText = document.createTextNode(pText);
    popupElement.appendChild(alertText);
    document.body.appendChild(popupElement);
    jspsAlignContainer(popupElement, pAlignment);
    window.setTimeout(jspsSPARemovePopupAlert, pTimeout * 1000)
  }

// Объявление служебной функции, удаляющей предупреждение
function jspsSPARemovePopupAlert()
  {
    var popupElement = document.all[JSPS_SPA_POPUP_ALERT_NAME];
    if (popupElement)
      document.body.removeChild(popupElement);
  }
```

Внимание!

Объявление функции `jspsShowPopupAlert` использует функцию `jspsAlignContainer`, чье объявление приведено в листинге 10.1. Также оно использует псевдоконстанты, объявленные в том же листинге.

Хорошая идея!

Поместите объявление этой функции в файл сценариев showpopupalert.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="showpopupalert.js"></SCRIPT>
```

Пример

Вот небольшая Web-страница, выводящая после загрузки предупреждение и "удерживающая" его на экран полсекунды.

```
<HTML>
  <HEAD>
    <TITLE>Страница, сообщающая об окончании загрузки</TITLE>
    <SCRIPT SRC="aligncontainer.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsAlignContainer
    помещено в файл сценария aligncontainer.js -->
    <SCRIPT SRC="showpopupalert.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsShowPopupAlert
    находится в файле сценариев showpopupalert.js -->
  </HEAD>
  <BODY>
    <P>Некоторое время прямо на этой Web-странице будет
    ↳отображаться сообщение об окончании ее загрузки.</P>
```

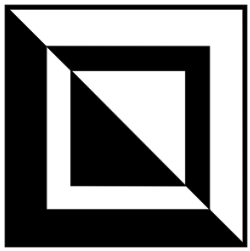
```
    <SCRIPT TYPE="text/javascript">
      jspsShowPopupAlert("Страница загружена", .5);
    </SCRIPT>
  </BODY>
</HTML>
```

Что дальше?

Ну вот, кое-что о вводе и выводе данных мы от народа узнали. Конечно, пока это самые простые приемы, но, как говорится, лиха беда начало.

Следующая глава будет посвящена, так сказать, смежной теме — сохранению и передаче данных от одной Web-страницы другой. Часто, особенно при написании Web-приложений, это приходится делать, а стандартных средств для сохранения данных и передачи их между страницами ни один Web-обозреватель не предоставляет. Но зато все Web-обозреватели поддерживают cookies и переход по гиперссылкам. Вот этим-то мы и воспользуемся (разумеется, под руководством народа)!

ГЛАВА 13



Сохранение данных и передача их другим Web-страницам

Web-обозреватели в силу их специфики не могут ни сохранять данные на компьютере клиента, ни напрямую передавать их другим Web-страницам. Точнее, они не предоставляют для этого стандартных средств, наподобие объекта `SaveData` или функции `sendDataToPage`. Но определенными способностями в этом плане они все-таки обладают, и грех ими не воспользоваться.

В этой главе народ будет "пытать" Web-обозреватели на предмет сохранения данных и прямой передачи их другим Web-страницам. А мы посмотрим, что из этого получится.

Как сохранить данные на клиентском компьютере?

Проблема

Мне очень нужно сохранить некоторые данные на компьютере клиента. Возможно ли это?

Решение

Возможно. Для этого мы используем особую возможность, поддерживаемую всеми Web-обозревателями и называемую cookies ("печенья"; единственное число — cookie).

Cookie — это небольшой фрагмент данных, сохраняемый Web-обозревателем на жестком диске клиентского компьютера (в отдельном текстовом файле, как Internet Explorer, или в одном большом "коммунальном" файле, как Firefox). Этот cookie может содержать, в принципе, любые данные, которые можно преобразовать в строковый формат (числа, логические величины, дату и, разумеется, строки). А это-то нам и нужно.

Сохраненный cookie доступен всем Web-страницам, хранящимся в одной папке Web-сервера и всех вложенных в нее папках. Поиск нужного cookie при обращении к нему выполняется Web-обозревателем автоматически; нам об этом заботиться не придется.

При создании cookie мы можем указать дату, до которой он будет храниться на диске клиентского компьютера. "Просроченные" cookie будут автоматически удалены самим Web-обозревателем. Также мы можем и не указывать дату хранения; в этом случае Web-обозреватель не будет сохранять cookie на диске — только в оперативной памяти *(временный cookie)*. Разумеется, в этом случае после закрытия Web-обозревателя все хранимые в памяти cookie будут потеряны.

На cookie налагаются следующие ограничения:

- максимальный размер — 4 Кбайт;
- максимальное количество на один домен (в смысле, на Web-сервер) — 20;
- максимальное общее количество, которые может хранить Web-обозреватель, — 300.

Думается, этих ограничений нам хватит. Мы же не содержимое Web-страниц в них будем сохранять, в конце концов!

Народ советует

И здесь автор прав. Не стоит хранить в cookie слишком уж большие данные.

Так как же создать cookie? Очень просто. Достаточно занести нужные данные в особом формате в свойство `cookie` объекта `document`. Остальное же, как говорится, дело техники.

Вот формат значения, заносимого в свойство `cookie` объекта `document`:

```
"<Имя значения>=<Само значение>[; expires=<Дата хранения>]
[; path=<Путь>][; domain=<Домен>][; secure]"
```

Имя значения задает имя, под которым заданное *Значение* будет храниться в cookie. Можно сказать, что в cookie для хранения каждого *Значения* создается переменная с заданным *Именем*.

А здесь нужно предупредить вот о чем. *Значение*, записываемое в cookie, не должно содержать пробелов, запятых и точек с запятой. Поэтому его перед занесением в cookie следует закодировать особым образом, для чего проще всего использовать стандартную функцию JavaScript `escape` (кстати, этот метод кодирования так и называется — *escape-кодирование*). Эта функция принимает единственный параметр — кодируемую строку — и возвращает ее в закодированном виде в качестве результата.

Дата хранения задает предельную дату, до которой cookie будет храниться на диске клиентского компьютера. Эта дата должна быть задана в строковом

формате и по Гринвичу (*GMT*, Grinwitch Meridian Time — время по гринвичскому меридиану). Для получения такой строки с датой в формате GMT следует воспользоваться не принимающим параметров методом `toGMTString` объекта `Date`. Если *Дата хранения* не задана, будет создан временный cookie (хранящийся в оперативной памяти компьютера).

Как уже говорилось, созданный в одной странице cookie доступен для всех страниц, хранящихся в той же папке Web-сервера и всех вложенных в нее папках. Если же мы хотим дать этому cookie доступ к страницам, хранящимся в другой папке Web-сервера (и вложенных в нее папках), то должны будем указать параметр *Путь*, ведущий к нужной папке. Например, значение `"/"` этого параметра делает cookie доступным для всех страниц на данном Web-сервере (это значение как раз и обозначает корневую папку сайта). А значение `"/chapters"` делает cookie доступным для страниц, сохраненных в папке chapters корневой папки сайта и всех вложенных в нее папках.

Если мы хотим дать доступ к cookie страницам, хранящимся на других Web-серверах, то должны будем также указать параметр *Домен*. Этот параметр должен представлять собой либо полное доменное имя Web-сервера без указания протокола, например:

www.othersite.ru

(тогда доступ к cookie получат страницы с Web-сервера **http://www.othersite.ru**), либо окончание доменного имени —

.othersite.ru

(тогда доступ к cookie получат страницы с Web-серверов **http://www.othersite.ru**, **http://www2.othersite.ru**, **http://foreign.othersite.ru** и др., в общем, те, чье доменное имя оканчивается на "othersite.ru"). Заметим, что начальная точка в окончании доменного имени обязательна.

Параметр `secure` своим присутствием указывает на то, что для доступа к данным, хранящимся в этом cookie, нужно использовать защищенное соединение, т. е. протокол *HTTPS* (HyperText Transfer Protocol Secured, защищенный протокол передачи гипертекста). Этот параметр используется только в том случае, если cookie создается запросом от Web-сервера — в общем, не наш случай.

Мы можем поместить в cookie сколько угодно значений с заданными именами, просто присваивая содержащие их строки в приведенном выше формате одну за другой свойству `cookie` объекта `document`. Все эти значения, сохраненные в cookie, будут существовать там, не "мешая" друг другу. Если одно из помещенных в cookie значений имеет то же имя, что и уже хранимое там, новое значение заменит старое (но остальные значения останутся нетронутыми).

Удалить какое-либо значение из cookie, не затрагивая остальных, судя по всему, невозможно. Зато мы можем удалить cookie целиком. Для этого достаточно сохранить в нем какое-либо произвольное значение с произвольным именем и в качестве даты хранения передать уже прошедшую дату, разумеется, в виде строки в формате GMT. Часто для этого используют строковое значение "Thu, 01-Jan-70 00:00:01 GMT", обозначающее одну секунду пополуночи первого января 1970 года (четверг). Получив такую дату, Web-обозреватель моментально "спохватится" и удалит этот cookie.

Чтобы получить из cookie сохраненные данные, достаточно, опять же, обратиться к свойству `cookie` объекта `document`. Оно вернет список всех сохраненных в cookie значений с их именами в виде пар *<Имя>=<Значение>*; эти пары будут отделены друг от друга точкой с запятой и пробелом.

Значения, возвращенные свойством `cookie`, будут закодированы с использованием метода кодирования `escape` — мы ведь кодировали их функцией `escape` перед присвоением этому свойству. За декодирование строкового значения "отвечает" стандартная функция JavaScript `unescape`, принимающая кодированную строку в качестве единственного параметра и возвращающая ее в декодированном виде.

Для облегчения работы с cookie можно использовать разработанные автором книги функции. Их описание будет приведено далее.

Функция `jspsSetCookie` (листинг 13.1) позволяет записать в cookie новое значение (или заменить старое). Формат ее вызова таков:

```
jspsSetCookie(<Имя значения>, <Само значение>[, <Дата хранения>
↪[, <Путь>[, <Домен>[, <Защищенный доступ>]]]]);
```

Комментировать тут особо нечего — все параметры этой функции были рассмотрены ранее, в разговоре о свойстве `cookie` объекта `document`. *Дата хранения* указывается в виде экземпляра объекта `Date`. *Защищенный доступ* указывается в логическом формате; значение `true` требует защищенного соединения для доступа к данным cookie, значение `false` — не требует. Значения остальных параметров указываются в строковом формате.

Значения функция `jspsSetCookie` не возвращает.

Функция `jspsGetCookie` (листинг 13.2) извлекает из cookie значения с заданным именем. Вот формат ее вызова:

```
jspsGetCookie(<Имя значения>);
```

Ну, тут уж совсем все понятно. Функция возвращает значение с заданным *Именем*, если оно существует в cookie: в противном случае возвращается `null`.

Функция `jspsDeleteCookie` (листинг 13.3) удаляет cookie. Формат ее вызова таков:

```
jspsDeleteCookie(<Имя значения>, [, <Путь>[, <Домен>]]);
```

Параметры этой функции нам также знакомы. Значения она не возвращает.

Народ предупреждает!

При использовании cookie нужно иметь в виду три вещи. Во-первых, посетитель может отключить в настройках безопасности своего Web-обозревателя поддержку cookie. Во-вторых, даже если поддержка cookie в Web-обозревателе включена, cookie может быть заблокирован самим Web-обозревателем или какой-либо сторонней программой, работающей совместно с ним, как небезопасный. В-третьих, современные программы для удаления шпионского программного обеспечения (spyware) могут также удалить cookie, посчитав его вредоносным. (Хотя, что такого может быть вредоносного в обычном куске текста — ума не приложу...)

Листинг 13.1. Функция `jspsSetCookie`, записывающая в cookie значение с указанным именем. Немного измененный вариант функции, приведенной в [2]

```
function jspsSetCookie(pName, pValue, pExpired, pPath, pDomain,
↳pIsSecured)
  {
    var s = pName + "=" + escape(pValue.toString());
    if (pExpired)
      s += "; expires=" + pExpired.toGMTString();
    if (pPath)
      s += "; path=" + pPath;
    if (pDomain)
      s += "; domain=" + pDomain;
    if (pIsSecured)
      s += "; secured";
    document.cookie = s;
  }
```

Листинг 13.2. Функция `jspsGetCookie`, считывающая из cookie значение с указанным именем. Немного измененный вариант функции, приведенной в [2]

```
function jspsGetCookie(pName)
  {
    var sCookie = document.cookie;
    var sCookieLength = sCookie.length;
```

```
    var sName = pName + "=";
    var sNameLength = sName.length;
    var sValue = null;
    var i = 0;
    var j = 0;
    while (i < sCookieLength)
      {
        j = i + sNameLength;
        if (sCookie.substring(i, j) == sName)
          {
            var n = sCookie.indexOf(";", j);
            if (n == -1) n = sCookieLength;
            sValue = unescape(sCookie.substring(j, n));
            break;
          }
        i = sCookie.indexOf(" ", i) + 1;
        if (i == 0) break;
      }
    return sValue;
  }
```

Листинг 13.3. Функция `jspsDeleteCookie`, удаляющая cookie. Немного измененный вариант функции, приведенной в [2]

```
function jspsDeleteCookie(pName, pPath, pDomain)
  {
    if (jspsGetCookie(pName))
      {
        var s = pName + "=; expires=Thu, 01-Jan-70 00:00:01 GMT";
        if (pPath)
          s += "; path=" + pPath;
        if (pDomain)
          s += "; domain=" + pDomain;
        document.cookie = s;
      }
  }
```

Хорошая идея!

Поместите объявления этих функций в файл сценариев cookies.js. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="cookies.js"></SCRIPT>
```

Народ советует

Функции, методы и объекты, расширяющие функциональность стандартных инструментов Web-обозревателя, — замечательные вещи. Они сильно ускоряют и облегчают написание сценариев. Так что лучше потратить несколько часов на написание таких функций, методов и объектов, чтобы потом выиграть, возможно, несколько часов, а то и дней.

Пример

Далее приведен пример Web-страницы с абзацем и набором кнопок, позволяющих задавать цвет текста этого абзаца. Выбранный с помощью этих кнопок цвет сохраняется в cookie. Кнопка **Сброс** позволяет удалить cookie и задать для текста цвет по умолчанию (красный).

```
<HTML>
  <HEAD>
    <TITLE>Страница, поддерживающая cookie</TITLE>
    <SCRIPT SRC="cookies.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с cookie
    находятся в файле сценариев cookies.js -->
    <SCRIPT SRC="datetimeutils.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы со значениями
    даты и времени находятся в файле сценариев datetimeutils.js -->
  </HEAD>
  <BODY>
    <P ID="par">Это содержимое страницы.</P>
    <SCRIPT TYPE="text/javascript">
      function colorize(pColor)
        {
          parColor = pColor;
          parObject.style.color = parColor;
          var dateExpired = new Date();
          dateExpired.addDays(1);
          jspsSetCookie("color", parColor, dateExpired);
        }

      var parObject = document.all["par"];
      var parColor = jspsGetCookie("color");
      if (!parColor) parColor = "#FF0000";
      colorize(parColor);
    </SCRIPT>
    <FORM>
      <INPUT TYPE="button" VALUE="Красный"
      ↳ONCLICK="colorize('#FF0000');">
```

```
      <INPUT TYPE="button" VALUE="Зеленый"
      ↳ONCLICK="colorize('#00FF00');">
      <INPUT TYPE="button" VALUE="Синий" ONCLICK="colorize('#0000FF');">
      <INPUT TYPE="button" VALUE="Сброс"
      ↳ONCLICK="jspsDeleteCookie('color');">
    </FORM>
  </BODY>
</HTML>
```

Внимание!

Приведенный пример использует метод `addDays` объекта `Date`, объявление которого было представлено в листинге 1.22.

В качестве даты хранения cookie мы задали завтрашний день. Для этого мы получили значение текущей даты и увеличили ее на один день с помощью метода `addDays` объекта `Date`, объявленного в листинге 1.22. Все это мы проделали в теле функции `colorize`, "раскрашивающей" текст абзаца в заданный цвет.

Если бы мы не задали дату хранения cookie, Web-обозреватель создал бы его в оперативной памяти, но не сохранил бы на диске. Последствия этого ясны — временный cookie был бы удален из памяти после завершения Web-обозревателя, и хранения данных на стороне клиента бы не получилось.

Как передать данные другой Web-странице?

Проблема

Каким образом можно передать данные от одной Web-страницы другой?

Решение 1

Ну, самое очевидное — использовать для этого cookie. Cookie мы уже рассмотрели, так что дополнительных объяснений не потребуется.

Как мы помним, cookie, созданные какой-либо страницей, доступны для всех страниц, хранящихся в той же папке Web-сервера и вложенных в нее папках. Если же мы хотим сделать cookie доступным для страниц, хранящихся в других папках, то должны будем задать параметр `path` в значении, присваиваемом свойству `cookie` объекта `document`, или соответствующий параметр функции `jspsSetCookie`. Собственно, об этом уже говорилось ранее.

Задавая параметру `domain` значение, присваиваемое свойству `cookie` объекта `document`, или соответствующий параметр функции `jspsSetCookie`, мы можем

дать доступ к cookie страницам, хранящимся на другом Web-сервере. Об этом также было сказано ранее, так что не будем повторяться.

Пример

Давайте возьмем Web-страницу из предыдущего примера (ту самую, с абзацем, меняющим цвет своего текста) и сохраним ее в файле jspsCookie.html. После этого создадим вторую страницу и сохраним ее в файле jspsCookieSecondary.html. HTML-код второй страницы приведен далее.

```
<HTML>
  <HEAD>
    <TITLE>Вторая страница, поддерживающая cookie</TITLE>
    <SCRIPT SRC="cookies.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с cookie
    находятся в файле сценариев cookies.js -->
  </HEAD>
  <BODY>
    <P ID="par">Это содержимое второй страницы.</P>
    <SCRIPT TYPE="text/javascript">
      var parObject = document.all["par"];
      var parColor = jspsGetCookie("color");
      if (!parColor) parColor = "#FF0000";
      parObject.style.color = parColor;
    </SCRIPT>
  </BODY>
</HTML>
```

Сначала откроем в Web-обозревателе страницу jspsCookie.html и зададим с помощью кнопок какой-либо цвет текста абзаца, отличный от цвета по умолчанию (красного). После этого откроем страницу jspsCookieSecondary.html. Мы увидим, что текст абзаца на этой странице принял тот же цвет, что и на странице jspsCookie.html. Это значит, что передача данных между страницами посредством cookie работает!

Решение 2

Использовать для передачи параметров интернет-адрес Web-страницы, добавив эти параметры в его конец. То есть фактически реализовать метод передачи данных GET, часто используемый в Web-программировании.

Давайте вспомним, что мы знаем о *методе передачи данных GET*. Возьмем, например, такой набор данных:

```
name1 = Ivan
surname = Ivanovich
```

```
name2 = Ivanov
age = 30
```

То есть здесь данные тоже объединяются в пары *<Имя>=<Значение>*, точно так же, как в случае с cookie.

Теперь подготовим приведенный набор данных для пересылки по методу GET (сами данные выделены полужирным шрифтом):

```
http://www.somesite.ru/bin/program.exe?
↳name1=Ivan&surname2=Ivanovich&name2=Ivanov&age=30
```

Видно, что пересылаемые по методу GET данные помещаются в самый конец интернет-адреса и отделяются от него вопросительным знаком. При этом пары *<Имя>=<Значение>* отделяются друг от друга знаком "коммерческое и" (&).

Данные, передаваемые методом GET, также должны кодироваться методом `escape`, дабы исключить из них пробелы, запятые, точки с запятой, русские буквы и некоторые другие символы, недопустимые в интернет-адресах. Для этого мы можем использовать уже знакомую нам функцию `escape`.

> ***Народ предупреждает!***
>
> Нужно обязательно иметь в виду, что полная длина интернет-адреса не должна превышать 255 символов. Так что передать очень большие данные методом GET мы не сможем.

Метод GET часто применяется для отправки данных серверным программам (программам, работающим на серверном компьютере совместно с Web-сервером). Но с таким же успехом мы можем пересылать с его помощью данные и другой Web-странице.

Что для этого нужно? Здесь есть два варианта...

- Создать обработчик события `onClick` гиперссылки, ведущей на другую страницу, и в этом обработчике добавлять к ее интернет-адресу передаваемые данные. При этом Web-обозреватель выполнит переход по уже измененному интернет-адресу, содержащему эти данные, которые и получит другая страница.
- Использовать для перехода на другую страницу свойство `href` объекта `location`, вложенного в объект `window`. Пожалуй, так будет проще: нам нужно только подготовить данные для передачи, добавить их к интернет-адресу другой страницы и присвоить полученный полный интернет-адрес упомянутому выше свойству.

Web-сценарий в странице, принимающей отправленные данные, должен выполнить вот что. Сначала он обратится к свойству `search` объекта `location`, вложенного в объект `window`; это свойство возвращает часть интернет-адреса, следующую за вопросительным знаком, вместе с самим этим знаком. После

этого он "разберет" полученные данные на пары *<Имя>=<Значение>* и декодирует их значения функцией `unescape`.

Для упрощения нашей задачи мы можем использовать две функции, описанные далее. Эти функции разработаны автором книги.

Функция `jspsGETEncodeData` (листинг 13.4) формирует на основе заданных данных строку для отправки методом GET. Вот формат ее вызова:

```
jspsGETEncodeData(<Хэш, содержащий данные>[, <Интернет-адрес>]);
```

Первым параметром этой функции передается хэш, содержащий данные. (Хэш, как мы уже знаем из *главы 1*, — массив, элементы которого имеют строковые, а не числовые индексы.) Индекс каждого элемента этого хэша станет *Именем* пары *<Имя>=<Значение>*, а его значение — *Значением* пары.

Вторым, необязательным, параметром передается *Интернет-адрес* (в смысле, интернет-адрес страницы, которой нужно передать данные). Если он указан, функция сама добавит к нему сформированную на основе содержимого *Хэша* строку, и мы получим полный адрес, который можно использовать для помещения в гиперссылку или свойство `href` объекта `location`, вложенного в объект `window`.

Функция `jspsGETEncodeData` возвращает либо сформированную на основе содержимого *Хэша* строку, уже с символом вопросительного знака, либо полный интернет-адрес, уже содержащий эту строку. Это зависит от того, был ли указан второй параметр этой функции.

Функция `jspsGETDecodeData` (листинг 13.5), наоборот, "разбирает" полученные данные и формирует на их основе хэш. Формат ее прост:

```
jspsGETDecodeData(<Пустой массив>);
```

Единственным параметром ей передается пустой, т. е. не содержащий элементов массив (если он все же содержит какие-то элементы, функция сама их удалит). На основе этого массива функция сформирует хэш, содержащий принятые данные. Значения она не возвращает.

Листинг 13.4. Функция `jspsGETEncodeData`, подготавливающая данные для пересылки методом GET

```
function jspsGETEncodeData(pDataArray, pHREF)
  {
    var s = "";
    var elIndex = "";
    for (elIndex in pDataArray)
      {
        if (s) s += "&";
```

```
        s += elIndex + "=" + escape(pDataArray[elIndex].toString());
      }
    if (s) s = "?" + s;
    if (pHREF)
      {
        var n = pHREF.indexOf("?");
        if (n != -1)
          pHREF = pHREF.substring(0, n - 1);
        s = pHREF + s;
      }
    return s;
  }
```

Листинг 13.5. Функция `jspsGETDecodeData`, декодирующая данные, переданные методом GET данной странице

```
function jspsGETDecodeData(pDataArray)
  {
    while (pDataArray.length > 0)
      pDataArray.pop();
    var s = window.location.search;
    var sLength = s.length;
    var i = 1;
    var j = 0;
    var k = 0;
    while (i < sLength)
      {
        j = s.indexOf("=", i);
        if (j == -1) break;
        var sName = s.substring(i, j);
        k = s.indexOf("&", j);
        if (k == -1) k = sLength;
        var sValue = s.substring(j + 1, k);
        pDataArray[sName] = unescape(sValue);
        i = k + 1;
      }
  }
```

Хорошая идея!

Поместите объявления этих функций в файл сценариев getdata.js. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="getdata.js"></SCRIPT>
```

Пример

Давайте создадим две Web-страницы. Первая страница будет запрашивать имя посетителя, для чего использует окно ввода данных, выводимое методом `prompt` объекта `window`. При щелчке по гиперссылке на этой странице откроется вторая страница, которая отобразит введенное посетителем имя.

Первая страница, реализующая ввод данных, будет иметь такой HTML-код:

```
<HTML>
  <HEAD>
    <TITLE>Первая страница, пересылающая данные</TITLE>
    <SCRIPT SRC="getdata.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с данными,
    передаваемыми методом GET, находятся в файле сценариев getdata.js -->
  </HEAD>
  <BODY>
    <H1>Запрос имени посетителя</H1>
    <SCRIPT TYPE="text/javascript">
      var userName = window.prompt("Введите свое имя", "");
      if (userName)
        {
          var dataArray = new Array();
          dataArray["username"] = userName;
          window.location.href = jspsGETEncodeData(dataArray,
          ↳"jspsGETDataSecondary.html");
        }
    </SCRIPT>
  </BODY>
</HTML>
```

Сохраним ее в файле jspsGETData.html.

Вторая страница, принимающая данные от первой, будет иметь вот такой код:

```
<HTML>
  <HEAD>
    <TITLE>Вторая страница, принимающая данные</TITLE>
    <SCRIPT SRC="getdata.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с данными,
    передаваемыми методом GET, находятся в файле сценариев getdata.js -->
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      var dataArray = new Array();
```

```
      jspsGETDecodeData(dataArray);
      document.write("<H1>Здравствуйте, " + dataArray["username"] +
      ↳"!</H1>");
    </SCRIPT>
  </BODY>
</HTML>
```

Ее мы сохраним в файле jspsGETDataSecondary.html.

Откроем в Web-обозревателе страницу jspsGETData.html. Введем в появившемся на экране диалоговом окне ввода Web-обозревателя свое имя и нажмем кнопку **OK** этого окна. Web-обозреватель тотчас загрузит страницу jspsGETDataSecondary.html, на которой будет красоваться введенное нами имя.

> ***Народ советует***
>
> Этот способ передачи данных (методом GET) заметно надежнее, чем предыдущий (с помощью cookie), поскольку, в отличие от cookie, интернет-адреса не блокируются сторонними программами. Так что мы можем быть спокойны — переданные нами методом GET данные дойдут до адресата в целости и сохранности.

Решение 3

Самое простое — прямой доступ к переменным страниц. Об этом мы уже говорили в *главе 4*, так что сейчас не будем повторяться.

Однако у этого решения есть три серьезных недостатка. Во-первых, таким образом могут "общаться" между собой только страница, открытая в созданном программно окне Web-обозревателя, и страница, которая это программно созданное окно создало. Во-вторых, обе эти страницы должны быть открыты одновременно (понятно, что в разных окнах Web-обозревателя). В-третьих, получить доступ к переменным страницы из другой страницы можно только после окончания ее загрузки, так что придется принимать дополнительные меры, дабы удостовериться, что страница загрузилась полностью.

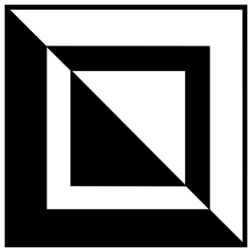
Поэтому такой подход для обмена данными между страницами применяют очень редко и практически всегда только в Web-приложениях.

Что дальше?

Вот и разобрались мы с сохранением данных на стороне клиента и пересылкой их между Web-страницами. Хоть Web-обозреватели и не позволяют это сделать, мы сумеем попросить их об этом.

А последняя глава в этой, четвертой, части будет посвящена работе с Web-формами. Вот там-то и начнется настоящая работа с данными, вводимыми посетителем!

ГЛАВА 14



Работа с Web-формами и элементами управления

Что ж, пора заняться настоящим вводом данных — с использованием Web-форм и элементов управления. Именно с их помощью осуществляется общение с посетителями на всех уважающих себя Web-сайтах (не всех, конечно, а только на тех, где ввод данных действительно нужен). Ведь Web-формы специально для этого и создаются, не так ли?

А еще мы поговорим о так называемых диалоговых окнах HTML, поддерживаемых неистощимым на подобные штуковины Internet Explorer. Диалоговые окна HTML могут очень пригодиться при создании корпоративных Web-решений.

Какие свойства, методы и события поддерживает Web-форма?

Проблема

Было бы неплохо получить список специфических свойств, методов и событий, поддерживаемых Web-формой.

Решение

Список всех специфических свойств, поддерживаемых Web-формой, приведен в табл. 14.1. Разумеется, экземпляр объекта, соответствующий форме, поддерживает и общие свойства всех элементов страниц, описанные в предыдущих главах (`id`, `style` и пр.).

Вот только вряд ли свойства формы будут нам полезны (за исключением, может быть, свойства `disabled`)... Куда более полезными для нас окажутся ее методы и события.

Таблица 14.1. *Специфические свойства формы*

Свойство	Описание
action	Задает интернет-адрес серверной программы, которой будут отправлены введенные в форму данные. Аналогично атрибуту ACTION тега <FORM>
disabled	Если false, форма доступна для ввода данных, если true, недоступна. Аналогично атрибуту DISABLED тега <FORM>. Поддерживается только Internet Explorer, начиная с версии 5.5
elements	Коллекция elements, содержащая все имеющиеся в форме элементы управления
enctype	Задает метод кодирования данных перед их отправкой. Аналогично атрибуту ENCTYPE тега <FORM>
method	Задает метод отправки данных. Аналогично атрибуту METHOD тега <FORM>
target	Задает цель формы, т. е. окно Web-обозревателя или фрейм, куда будет выведена Web-страница — результат работы серверной программы. Аналогично атрибуту TARGET тега <FORM>

Форма поддерживает два специфических метода. Метод submit запускает отправку данных серверной программе и аналогичен нажатию кнопки **Отправить**. Метод reset очищает форму и подставляет в ее элементы управления заданные в коде HTML формы значения по умолчанию; этим он аналогичен нажатию кнопки **Сброс**. Оба этих метода не принимают параметров и не возвращают результата.

Событий, поддерживаемых формой, тоже два. Событие onSubmit наступает перед отправкой формой данных серверной программе, вызываемой нажатием кнопки **Отправить** или вызовом метода submit. Событие onReset же наступает перед очисткой формы, которая выполняется после нажатия кнопки **Сброс** или вызова метода reset.

Мы можем прервать отправку данных серверной программе или очистку формы, вернув из обработчика события onSubmit или onReset соответственно значение false (для этого, как мы помним из *главы 2*, нужно использовать выражение return false). Это очень удобно, если мы хотим выполнить проверку введенных данных и отменить их отправку, если в форму введены некорректные данные, либо подставить в очищаемую форму какие-то другие значения, отличные от заданных в HTML-коде формы.

Народ советует

Действительно, стоит взять на заметку событие onSubmit. Сценарии, осуществляющие проверку введенных в форму данных на корректность, практически

всегда выполняются в виде обработчиков этого события. Если сценарий сочтет введенные данные некорректными, он вернет значение `true`, и отправка данных будет выполнена. В противном случае он вернет значение `false`, и форма не станет отправлять данные. Тогда полезно предусмотреть какое-либо предупреждение для посетителя, говорящее о некорректности введенных им данных.

Какие свойства, методы и события поддерживают элементы управления?

Проблема

А что там насчет свойств, методов и событий элементов управления, предусматриваемых стандартом HTML? Они, пожалуй, будут для меня более полезны...

Решение

Давайте сначала перечислим все элементы управления, описанные в стандарте HTML. Как мы знаем, большинство их создается одинарным тегом `<INPUT>` и отличается значением атрибута `TYPE` этого тега. Все доступные значения атрибута `TYPE` тега `<INPUT>` приведены в табл. 14.2.

Таблица 14.2. *Значения атрибута `TYPE` тега `<INPUT>`, с помощью которого создаются элементы управления*

Значение	Описание
`button`	Обычная кнопка
`checkbox`	Флажок
`file`	Поле ввода имени файла с кнопкой **Открыть**. Позволяет отправить серверной программе файл
`hidden`	Скрытое поле. Никак не отображается на странице, но его значение отправляется серверной программе вместе с остальными данными, введенными в форму. Может использоваться, например, для уникальной идентификации посетителя или передачи служебных данных
`image`	Кнопка-изображение. Аналогична по действию кнопке `submit` *(см. далее)*
`password`	Поле ввода пароля. Вводимый текст отображается в виде звездочек или точек, в зависимости от версии Windows
`radio`	Переключатель
`reset`	Кнопка, при нажатии которой происходит очистка формы (элементы управления принимают значения, определенные в коде HTML формы как значения по умолчанию). Называется также кнопкой **Сброс** или кнопкой `reset`

Таблица 14.2 *(окончание)*

Значение	Описание
submit	Кнопка, при нажатии которой происходит отправка данных, введенных в форму, серверной программе. Называется также кнопкой **Отправить** или кнопкой submit
text	Поле ввода

Область редактирования текста создается с помощью парного тега <TEXTAREA>. Внутри этого тега помещается содержимое области редактирования по умолчанию.

Список, обычный или раскрывающийся, создается с помощью парного тега <SELECT>. Внутри этого тега помещаются парные теги <OPTION>, создающие пункты этого списка. Название каждого пункта помещается внутрь соответствующего ему тега <OPTION>.

В табл. 14.3 приведены все свойства, поддерживаемые элементами управления. Элементы управления, поддерживающие эти свойства, указаны либо в виде значения атрибута TYPE тега <INPUT>, либо в виде создающего их тега.

Таблица 14.3. *Специфические свойства элементов управления*

Свойство	Элемент управления	Описание
checked	checkbox, radio	Задает состояние флажка или переключателя. Если равно true, флажок или переключатель включен, если false, отключен
cols	<TEXTAREA>	Задает ширину области редактирования в символах. Аналогично атрибуту COLS тега <TEXTAREA>
defaultChecked	checkbox, radio	Возвращает состояние флажка или переключателя по умолчанию, заданное в создающем его коде HTML (true, если флажок или переключатель включен, false, если отключен). Аналогично атрибуту CHECKED тега <INPUT>
defaultSelected	<OPTION>	Возвращает true, если данный пункт должен быть выбран в списке по умолчанию (в смысле, так определено в создающем его коде HTML), и false в противном случае. Аналогично атрибуту SELECTED тега <OPTION>
defaultValue	file, password, text	Возвращает значение по умолчанию поля ввода, заданное в создающем его коде HTML. Аналогично атрибуту VALUE тега <INPUT>

Таблица 14.3 *(продолжение)*

Свойство	Элемент управления	Описание
disabled	Все	Если false, элемент управления доступен для ввода данных, если true, недоступен. Аналогично атрибуту DISABLED соответствующего тега <INPUT>
form	Все	Возвращает форму, в которой находится данный элемент управления
indeterminate	checkbox	Если true, флажок находится в неопределенном состоянии (закрашен серым), если false — в состоянии, заданном свойством checked. Поддерживается только Internet Explorer, начиная с версии 4.0
index	<OPTION>	Возвращает номер пункта в списке. Нумерация пунктов начинается с нуля
length	<SELECT>	Возвращает количество пунктов списка
maxLength	password, text	Задает максимальное количество символов, которое можно ввести в поле ввода. Аналогично атрибуту MAXLENGTH тега <INPUT>
multiple	<SELECT>	Если true, посетитель может выбрать в списке сразу несколько пунктов, если false, то только один. Аналогично атрибуту MULTIPLE тега <SELECT>
options	<SELECT>	Коллекция options, содержащая все пункты списка
readOnly	password, text	Если true, поле ввода доступно только для чтения, если false — и для чтения, и для записи текста. Аналогично атрибуту READONLY тега <INPUT>
rows	<TEXTAREA>	Задает высоту области редактирования в строках. Аналогично атрибуту ROWS тега <TEXTAREA>
selected	<OPTION>	Если true, данный пункт выбран в списке, если false, то не выбран
selectedIndex	<SELECT>	Номер выбранного посетителем пункта списка. Нумерация пунктов начинается с нуля
size	<SELECT>	Задает количество одновременно выводимых в списке пунктов. Если равно 1, создается раскрывающийся список, если больше 1 — обычный список. Аналогично атрибуту SIZE тега <SELECT>

Таблица 14.3 *(окончание)*

Свойство	Элемент управления	Описание
tabIndex	Все	Задает номер в порядке обхода при нажатии клавиши <Tab> (прямой порядок) или комбинации клавиш <Shift>+<Tab> (обратный порядок). Аналогично атрибуту TABINDEX тега <INPUT>
text	<OPTION>	Задает текст пункта списка
type	Все, создаваемые тегом <INPUT>	Возвращает тип элемента управления, заданный атрибутом TYPE тега <INPUT>. Для области редактирования возвращает "textarea", для обычного списка — "select-multiple", для раскрывающегося списка — "select-one"
value	Все	Значение элемента управления, которое он отправит серверной программе (либо текст, введенный в поле ввода или область редактирования, либо значение, заданное атрибутом VALUE тега <INPUT> или <OPTION>)
wrap	<TEXTAREA>	Задает способ разрыва строк. Доступны значения: "off" (строки не разрываются; значение по умолчанию), "soft" (просто разрывает строки, не внося в них изменений) и "hard" (не только разрывает строки, но и вставляет в местах разрыва символы возврата каретки и перевода строки). Аналогично атрибуту WRAP тега <TEXTAREA>

Кроме того, элементы управления поддерживают все "обычные" свойства, присущие остальным элементам страницы. Это уже знакомые нам свойства id, style и др.

Список методов, поддерживаемый элементами управления, не столь велик. Их всего три. Метод blur убирает фокус ввода с элемента управления, метод focus, наоборот, устанавливает фокус на элемент управления. А метод select, поддерживаемый только полями ввода, выделяет все содержимое поля ввода. Все эти методы не принимают параметров и не возвращают результата.

Поддерживаемых элементами управления событий тоже немного. Все они перечислены в табл. 14.4.

Таблица 14.4. *Специфические события элементов управления*

Событие	Элемент управления	Описание
onBlur	Все	Наступает, когда текущий элемент управления теряет фокус ввода

Таблица 14.4 *(окончание)*

Событие	Элемент управления	Описание
onChange	password, text	Наступает после того, как посетитель изменит данные в поле ввода и переместит фокус ввода на другой элемент управления. Предшествует событию onBlur
onClick	button, checkbox, radio, reset, submit	Наступает, когда пользователь щелкает по кнопке, флажку или переключателю, и кнопка срабатывает, а флажок или переключатель меняет свое состояние
onFocus	Все	Наступает, когда текущий элемент управления получает фокус ввода

Простейшие манипуляции с формами и элементами управления

Начнем с самого простого: как добраться до нужной формы или элемента управления, как извлечь введенный в поле ввода текст или узнать состояние флажка, как подставить в поле ввода значение, вычисленное в сценарии, как временно сделать элемент управления недоступным, как заполнить программно список и пр.

Как получить доступ к нужной форме?

Проблема

Я могу получить доступ к форме так же, как и к любому другому элементу страницы — через коллекцию all объекта document, — или же для этого нужно использовать что-то иное?

Решение

Да, для этого можно использовать коллекцию all объекта document. В конце концов, эта коллекция замышлялась как универсальный, "всеобъемлющий" инструмент для доступа к любому элементу страницы, будь то абзац, таблица, графическое изображение, элемент ActiveX или форма.

Кроме того, объект document поддерживает коллекцию forms. Она содержит все созданные на странице формы. (Собственно, о ней уже упоминалось в *главе 2*, но, как говорится, повторение — мать учения.)

Народ советует

Пожалуй, для получения доступа к формам удобнее использовать именно коллекцию forms объекта document. Эта коллекция обычно содержит значительно

меньше элементов, чем "глобальная" коллекция `all`, поэтому поиск в ней выполняется быстрее.

Народ предупреждает!

Мы уже давно знаем, что имя элемента страницы задается с помощью атрибута `ID` создающего этот элемент тега. Но в случае форм и элементов управления их имена нужно указать также в атрибуте `NAME`:

```
<FORM ID="frm" NAME="frm">
```

Если этого не сделать, форма и ее элементы управления могут работать неправильно.

Пример

```
<FORM ID="frm" NAME="frm">
  <!-- Здесь помещаются элементы управления -->
</FORM>
<SCRIPT TYPE="text/javascript">
  var frmObject = document.forms["frm"];
</SCRIPT>
```

Приведенный фрагмент HTML-кода содержит сценарий, помещающий в переменную `frmObject` форму `frm`.

Как получить доступ к нужному элементу управления в форме?

Проблема

А для получения доступа к элементу управления, находящемуся в форме, мне тоже нужно использовать коллекцию `all` объекта `document`?

Решение

Можно, конечно, но необязательно. Любая форма поддерживает коллекцию `elements`, содержащую все созданные в ней элементы управления. Если нужно получить доступ сразу к нескольким элементам управления, созданным в одной форме, можно сначала получить доступ к самой этой форме, а потом уже через коллекцию `elements` "добраться" до ее элементов управления.

Пример

```
<FORM ID="frm" NAME="frm">
  Имя посетителя:
  <INPUT TYPE="text" ID="name" NAME="name">
  Пароль посетителя:
  <INPUT TYPE="password" ID="password" NAME="password">
```

```
  <INPUT TYPE="submit" ID="submit" NAME="submit" VALUE="Войти на сайт">
</FORM>
<SCRIPT TYPE="text/javascript">
  var frmObject = document.forms["frm"];
  var nameObject = frmObject.elements["name"];
  var passwordObject = frmObject.elements["password"];
  var submitObject = frmObject.elements["submit"];
  nameObject.disabled = true;
  passwordObject.disabled = true;
  submitObject.disabled = true;
</SCRIPT>
```

Приведенный фрагмент HTML-кода содержит сценарий, запрещающий доступ ко всем элементам управления в форме `frm`.

Как получить значение элемента управления?

Проблема

Как я могу получить значение элемента управления, например, текст, введенный в поле ввода, или состояние флажка?

Решение

Проще всего с полями ввода и областями редактирования — чтобы получить их содержимое, нужно обратиться к свойству `value`. Это свойство вернет текст, введенный в поле ввода или область редактирования, в строковом виде.

> ***Народ предупреждает!***
>
> Текст, введенный в поле ввода, всегда возвращается свойством `value` в строковом виде. Если же в данное поле ввода должно вводиться, скажем, число, нам придется преобразовать его в числовой формат, чтобы использовать в вычислениях. Как это сделать, было описано еще в *главе 1*.

Состояние флажка можно узнать, обратившись к свойству `checked`. Это свойство возвращает `true`, если флажок включен, и `false`, если он отключен.

С переключателями дело обстоит несколько сложнее, т. к. переключатели никогда не используются поодиночке, а только группами (иначе в них нет смысла). Все переключатели, входящие в группу, имеют одинаковое имя, т. е. значение атрибутов `ID` и `NAME`. Поэтому мы не сможем просто так обратиться к какому-либо одному переключателю по его имени и узнать, включен ли он, — мы "попадем" на всю группу.

Пожалуй, единственный способ выяснить состояние переключателя — это "перебрать" все элементы коллекции `elements` формы и найти все переключа-

тели нужной группы. Сделать это несложно — как мы помним, значение свойства `type` у переключателя всегда равно `"radio"`, а имя его группы можно узнать, обратившись к свойствам `id` или `name`. А уж, найдя переключатели заданной группы, мы можем без труда узнать их состояние, обратившись к свойству `checked` — если переключатель включен, оно будет содержать значение `true`.

Создав переменную, присвоив ей 0 и увеличивая ее значение с каждым "найденным" переключателем нужной группы, мы без труда сможем выяснить номер включенного переключателя в группе — он будет содержаться в этой переменной. Мы также можем получить значение свойства `value` включенного переключателя, т. е. его значение.

Написанная автором книги функция `jspsGetCheckedRadioIndex` (листинг 14.1) позволяет выяснить номер включенного переключателя в группе. Формат ее вызова крайне прост:

```
jspsGetCheckedRadioIndex(<Форма>, <Имя группы переключателей>);
```

Первым параметром передается форма, в которой находится группа переключателей, вторым — имя этой группы переключателей в строковом виде.

Функция `jspsGetCheckedRadioIndex` возвращает номер включенного переключателя в числовом виде. Если ни один переключатель группы не включен, возвращается значение –1.

Листинг 14.1. Функция `jspsGetCheckedRadioIndex`, возвращающая номер включенного переключателя заданной группы. Основана на коде функции, приведенной в [1]

```
function jspsGetCheckedRadioIndex(pForm, pGroupName)
  {
    var radioIndex = -1;
    var i = 0;
    var formElement = null;
    for (i = 0; i < pForm.elements.length; i++)
      {
        var formElement = pForm.elements[i];
        if ((formElement.type == "radio") &&
        ↳(formElement.id == pGroupName))
          {
            radioIndex++;
            if (formElement.checked) break;
          }
      }
    return radioIndex;
  }
```

А функция jspsGetCheckedRadioValue (листинг 14.2) позволяет выяснить значение включенного переключателя, заданное атрибутом VALUE тега <INPUT>. Формат ее вызова также прост:

```
jspsGetCheckedRadioValue(<Форма>, <Имя группы переключателей>);
```

Параметры здесь те же, что и у функции jspsGetCheckedRadioIndex, так что не будем на них останавливаться.

Функция jspsGetCheckedRadioValue возвращает значение включенного переключателя в строковом виде. Если ни один переключатель группы не включен, возвращается значение null.

Листинг 14.2. Функция jspsGetCheckedRadioValue, возвращающая значение включенного переключателя заданной группы. Немного измененный вариант функции, приведенной в [1]

```
function jspsGetCheckedRadioValue(pForm, pGroupName)
  {
    var radioValue = null;
    var i = 0;
    var formElement = null;
    for (i = 0; i < pForm.elements.length; i++)
      {
        var formElement = pForm.elements[i];
        if ((formElement.type == "radio") &&
        ↳(formElement.id == pGroupName))
          if (formElement.checked)
            {
              radioValue = formElement.value;
              break;
            }
      }
    return radioValue;
  }
```

Со списками в этом смысле несколько проще. Список поддерживает свойство selectedIndex, возвращающее номер выбранного пункта. (Не забываем, что нумерация пунктов списка начинается с нуля.) Этот номер возвращается в числовом формате, так что мы сразу сможем использовать его, например, в выражениях сравнения. Если ни один из пунктов в списке не выбран, свойство selectedIndex возвращает значение –1.

Воспользовавшись коллекцией options, поддерживаемой списком и содержащей все его пункты, мы можем выяснить значение выбранного в списке

пункта. Это значение, как мы знаем, задается атрибутом VALUE тега <OPTION>. Сценарий, позволяющий узнать значение выбранного пункта списка, очень прост:

```
var lstObject = document.all["lst"];
var optionIndex = lstObject.selectedIndex;
if (optionIndex == -1)
  var optionValue = lstObject.options[optionIndex].value
```

Сначала мы проверяем, выбран ли хоть один пункт из списка (равно ли –1 значение свойства selectedIndex). Если же в списке был выбран какой-то пункт, мы сначала получаем его, воспользовавшись коллекцией options списка, а потом без труда извлекаем его значение.

Но что делать, если мы создали список, в котором посетитель может выбрать одновременно несколько пунктов? (Для этого достаточно включить в тег <SELECT> не имеющий значения атрибут MULTIPLE, который уже одним своим присутствием в теге дает списку эту возможность.) Ничего не поделаешь — придется просматривать всю коллекцию options этого списка и искать все пункты, значение свойства selected которых равно true.

Функция jspsGetSelectedOptionIndices (листинг 14.3) позволяет выяснить, какие пункты списка выбраны. Вот формат ее вызова:

```
jspsGetSelectedOptionIndices(<Имя списка>);
```

Единственным параметром этой функции передается имя списка в строковом виде. Функция jspsGetSelectedOptionIndices возвращает массив, содержащий номера выбранных в списке пунктов в числовом виде. Если ни один из пунктов списка не выбран, возвращается значение null.

Листинг 14.3. Функция jspsGetSelectedOptionIndices, возвращающая номера выбранных в списке пунктов

```
function jspsGetSelectedOptionIndices(pSelectName)
  {
    var selectObject = document.all[pSelectName];
    var optionIndices = null;
    var i = 0;
    for (i = 0; i < selectObject.options.length; i++)
      if (selectObject.options[i].selected)
        {
          if (optionIndices == null)
            optionIndices = new Array();
          optionIndices[optionIndices.length] = i;
        }
```

```
    return optionIndices;
  }
```

А функция `jspsGetSelectedOptionValues` (листинг 14.4) позволяет узнать значения выбранных в списке пунктов. Вот формат ее вызова:

```
jspsGetSelectedOptionValues(<Имя списка>);
```

Опять же, в эту функцию передается имя списка в строковом виде. Возвращает она массив, содержащий значения выбранных в списке пунктов в строковом виде. Если ни один пункт не выбран, также возвращается значение `null`.

Листинг 14.4. Функция `jspsGetSelectedOptionValues`, возвращающая значения выбранных в списке пунктов

```
function jspsGetSelectedOptionValues(pSelectName)
  {
    var selectObject = document.all[pSelectName];
    var optionValues = null;
    var i = 0;
    var optionObject = null;
    for (i = 0; i < selectObject.options.length; i++)
      {
        var optionObject = selectObject.options[i];
        if (optionObject.selected)
          {
            if (optionValues == null)
              optionValues = new Array();
            optionValues[optionValues.length] = optionObject.value;
          }
      }
    return optionValues;
  }
```

Хорошая идея!

Поместите объявления всех этих функций в файл сценариев controls.js. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="controls.js"></SCRIPT>
```

Пример 1

Далее приведен HTML-код Web-страницы с формой. При нажатии кнопки **Получить данные** на экране появится окно-сообщение, отображающее введенные в форму данные.

```
<HTML>
  <HEAD>
    <TITLE>Страница с формой</TITLE>
    <SCRIPT SRC="controls.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с элементами
    управления находятся в файле сценариев controls.js -->
    <SCRIPT TYPE="text/javascript">
      function btnOnClick()
        {
          var frmObject = document.forms["frm"];
          var nameObject = frmObject.elements["name"];
          var emailObject = frmObject.elements["email"];
          var qualObject = frmObject.elements["qual"];
          var maillistObject = frmObject.elements["maillist"];
          s = "name: " + nameObject.value + "\n";
          s += "email: " + emailObject.value + "\n";
          s += "qual: " + jspsGetCheckedRadioValue(frmObject, "qual") +
          ↳"\n";
          s += "maillist: " + (maillistObject.checked ? "включен" : "");
          window.alert(s);
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Имя:
      <INPUT TYPE="text" ID="name" NAME="name"><BR><BR>
      E-mail:
      <INPUT TYPE="text" ID="email" NAME="email"><BR><BR>
      Квалификация:<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="novice">
      Новичок<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="advanced">
      Опытный<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="guru">
      Гуру<BR><BR>
      Подписаться на рассылку:
      <INPUT TYPE="checkbox" ID="maillist" NAME="maillist" CHECKED>
      ↳<BR><BR><BR>
      <INPUT TYPE="button" ID="btn" NAME="btn" VALUE="Получить данные"
      ↳ONCLICK="btnOnClick();">
    </FORM>
  </BODY>
</HTML>
```

Обратим внимание, как в теле функции-обработчика события `onClick` кнопки `btn` (она имеет имя `btnObClick`) мы использовали объявленную в листинге 14.2 функцию `jspsGetCheckedRadioValue`, чтобы выяснить значение включенного переключателя группы `qual`.

Народ советует

Во многих случаях бывает нужнее номер включенного переключателя — его удобнее использовать в выражениях сравнения.

Пример 2

Далее приведен пример другой Web-страницы с формой. На этот раз форма содержит список с возможностью выбора сразу нескольких пунктов. При нажатии кнопки **Получить данные** на экране появится окно-сообщение, представляющее значения всех выбранных в списке пунктов.

```
<HTML>
  <HEAD>
    <TITLE>Страница с формой</TITLE>
    <SCRIPT SRC="controls.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с элементами
    управления находятся в файле сценариев controls.js -->
    <SCRIPT TYPE="text/javascript">
      function btnOnClick()
        {
          var frmObject = document.forms["frm"];
          var optionArray = jspsGetSelectedOptionValues("prefs");
          var s = "prefs: "
          if (optionArray)
            for (var i = 0; i < optionArray.length; i++)
              s += optionArray[i].toString() + " ";
          window.alert(s);
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Ваши любимые языки программирования:
      <SELECT ID="prefs" NAME="prefs" SIZE="4" MULTIPLE>
        <OPTION VALUE="javascript">JavaScript</OPTION>
        <OPTION VALUE="java">Java</OPTION>
        <OPTION VALUE="cpp">C++</OPTION>
        <OPTION VALUE="delphi">Borland Delphi</OPTION>
        <OPTION VALUE="vb">Microsoft Visual Basic</OPTION>
      </SELECT>
```

```
      <INPUT TYPE="button" ID="btn" NAME="btn" VALUE="Получить данные"
      ↳ONCLICK="btnOnClick();">
    </FORM>
  </BODY>
</HTML>
```

Народ советует

Списки с возможностью одновременного выбора нескольких пунктов применяют весьма редко, т. к. они не очень наглядны и могут вызвать затруднения у неопытных посетителей. Практически всегда, чтобы дать возможность посетителю выбрать сразу несколько альтернатив (или не выбрать ни одной), применяют наборы флажков. С флажками меньше проблем и их зачастую проще обрабатывать в Web-сценариях и серверных программах.

Пожалуй, списки с возможностью одновременного выбора нескольких пунктов могут пригодиться только в том случае, если набор альтернатив, которые мы хотим предложить посетителю, слишком велик, и соответствующий набор флажков занял бы чересчур много места на странице. Тогда лучше использовать список — он компактнее.

Как программно установить новое значение элемента управления?

Проблема

Очень часто приходится устанавливать новое значение элемента управления программно, из Web-сценария. Как это сделать проще всего?

Решение

Да, собственно, ничего сложного тут нет. Достаточно воспользоваться соответствующим свойством этого элемента управления.

- Для полей ввода и областей редактирования это будет свойство `value`. Новое значение поля ввода или области редактирования присваивается ему в строковом виде.
- Для флажков это будет свойство `checked`. Оно имеет логический тип; значение `true` делает флажок включенным, значение `false` — отключенным.
- Для переключателей это также будет свойство `checked`. Правда, как мы уже знаем, здесь не все так просто, о чем и пойдет речь далее.
- Для списков с возможностью выбора только одного пункта это будет свойство `selectedIndex`. Этому свойству присваивается номер выбранного пункта списка.
- Для списков с возможностью выбора нескольких пунктов одновременно это будет свойство `selected` каждого пункта, который должен быть вы-

бран. Свойство `selected` имеет логический тип: значение `true` делает пункт списка выбранным, значение `false` — невыбранным. Далее мы поговорим о списках подробнее.

Разумеется, чтобы добраться до нужного элемента управления, мы должны задать для него имя. Не забываем, что имя элемента управления должно быть задано и атрибутом `ID`, и атрибутом `NAME`, иначе элемент управления будет работать неправильно или вообще не заработает.

Теперь давайте займемся переключателями. Да, группы переключателей — самый проблемный элемент управления в Web-форме. Мы не можем обратиться напрямую по имени к нужному нам переключателю, т. к. вся группа переключателей носит одно имя. Что же делать?

Пожалуй, есть только один способ добраться до нужного переключателя — перебором всех элементов управления, находящихся в форме. По крайней мере, другие автору книги не известны. Искать же нужный нам переключатель мы можем как по его номеру в группе, так и по его значению.

Итак, чтобы найти нужный переключатель по его номеру в группе, мы должны выполнить последовательность действий, приведенную далее.

1. Объявляем переменную и присваиваем ей значение 0. Эта переменная будет служить счетчиком переключателей.
2. Выбираем из коллекции `elements` формы очередной элемент управления.
3. Проверяем, переключатель ли это (содержит ли его свойство `type` значение `"radio"`) и то ли он имеет имя (значение свойства `id`). Если это переключатель с нужным нам именем, увеличиваем значение объявленной на шаге 1 переменной на единицу.
4. Сравниваем значение объявленной на шаге 1 переменной с нужным нам номером переключателя. Если они равны (т. е. мы нашли переключатель с нужным номером), то завершаем работу.
5. Если это не последний элемент управления формы, переходим к шагу 2.

Что касается поиска переключателя по его значению, то тут дело обстоит много проще. Нам будет достаточно просмотреть коллекцию `elements` формы, выбрать оттуда все переключатели (свойство `type` содержит значение `"radio"`) с заданным именем (свойство `id`) и найти нужный по значению свойства `value`.

А уж получив нужный нам переключатель, мы можем включить его или отключить. Для этого, как мы уже знаем, предназначено свойство `checked`. Присвоенное этому свойству значение `true` включает переключатель, значение `false` отключает его.

Автор книги, известный лентяй, написал функцию `jspsGetRadioByIndex` (листинг 14.5), позволяющую получить доступ к переключателю по его номеру в группе. Формат ее вызова таков:

```
jspsGetRadioByIndex(<Форма>, <Имя группы переключателей>,
↳<Номер переключателя>);
```

Первым параметром передается форма, в которой находится группа переключателей, вторым — имя этой группы переключателей в строковом виде, третьим — номер искомого переключателя в группе в числовом виде. Не забываем, что переключатели в группе нумеруются, начиная с нуля.

Функция `jspsGetRadioByIndex` возвращает найденный переключатель. Если группа не содержит переключателя с таким номером, возвращается значение `null`.

Листинг 14.5. Функция `jspsGetRadioByIndex`, возвращающая переключатель заданной группы с заданным номером

```
function jspsGetRadioByIndex(pForm, pGroupName, pIndex)
  {
    var radioIndex = -1;
    var i = 0;
    var formElement = null;
    for (i = 0; i < pForm.elements.length; i++)
      {
        var formElement = pForm.elements[i];
        if ((formElement.type == "radio") &&
        ↳(formElement.id == pGroupName))
          {
            radioIndex++;
            if (radioIndex == pIndex) break;
          }
      }
    return formElement;
  }
```

Функция `jspsGetRadioByValue` (листинг 14.6) позволяет "добраться" до переключателя с известным значением. Вот формат ее вызова:

```
jspsGetRadioByValue(<Форма>, <Имя группы переключателей>,
<Значение переключателя>);
```

Первым параметром передается форма, в которой находится группа переключателей, вторым — имя этой группы переключателей в строковом виде, третьим — значение искомого переключателя также в строковом виде.

Функция `jspsGetRadioByValue` возвращает найденный переключатель. Если группа не содержит переключателя с таким номером, возвращается значение `null`.

Листинг 14.6. Функция `jspsGetRadioByValue`, возвращающая переключатель заданной группы с заданным значением

```
function jspsGetRadioByValue(pForm, pGroupName, pValue)
  {
    var i = 0;
    var formElement = null;
    for (i = 0; i < pForm.elements.length; i++)
      {
        var formElement = pForm.elements[i];
        if ((formElement.type == "radio") &&
        ↳(formElement.id == pGroupName))
          if (formElement.value == pValue) break;
      }
    return formElement;
  }
```

Хорошая идея!

Добавьте объявление этих функций в файл сценариев controls.js. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="controls.js"></SCRIPT>
```

Что касается списков с возможностью одновременного выбора нескольких пунктов, то с ними все проще, особенно если нам известен номер нужного пункта. Тогда мы используем выражение вида

`<`*`Список`*`>.options[<`*`Номер пункта`*`>].selected = <`*`Выбран или не выбран`*`>;`

для всех пунктов, которые нам нужно сделать выбранными. Значение `true` свойства `selected` выбирает данный пункт, а значение `false` делает его невыбранным.

Если нам известно значение элемента, который нужно сделать выбранным, то нам придется просмотреть в его поисках всю коллекцию `options` списка. Увы — другого способа нет!

Пример 1

Далее приведен HTML-код Web-страницы с формой. При загрузке этой страницы особый сценарий подставляет в элементы управления формы изначальные значения.

```
<HTML>
  <HEAD>
    <TITLE>Страница с формой</TITLE>
    <SCRIPT SRC="controls.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с элементами
    управления находятся в файле сценариев controls.js -->
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Имя:
      <INPUT TYPE="text" ID="name" NAME="name"><BR><BR>
      E-mail:
      <INPUT TYPE="text" ID="email" NAME="email"><BR><BR>
      Квалификация:<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="novice">
      ↳Новичок<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="advanced">
      ↳Опытный<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="guru">
      ↳Гуру<BR><BR>
      Подписаться на рассылку:
      <INPUT TYPE="checkbox" ID="maillist" NAME="maillist"><BR><BR><BR>
      <INPUT TYPE="submit" ID="btn" NAME="btn">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      var frmObject = document.forms["frm"];
      var nameObject = frmObject.elements["name"];
      var emailObject = frmObject.elements["email"];
      var guruObject = jspsGetRadioByValue(frmObject, "qual", "guru");
      var maillistObject = frmObject.elements["maillist"];
      nameObject.value = "Superuser";
      emailObject.value = "superuser@supermailserver.ru";
      guruObject.checked = true;
      maillistObject.checked = true;
    </SCRIPT>
  </BODY>
</HTML>
```

Обратим внимание, как мы делаем третий переключатель группы `qual` включенным. Сначала мы получаем его с помощью функции `jspsGetRadioByValue`, которой передаем его значение — `"guru"`. (Конечно, мы можем найти его и по номеру — 2 — с помощью функции `jspsGetRadioByIndex`, но это не очень на-

глядно.) Получив этот переключатель, мы присваиваем его свойству `checked` значение `true`. Все — переключатель включен!

> ***Народ советует***
>
> Исходные значения для элементов управления формы лучше задавать в коде HTML — так проще и нагляднее (если, конечно, эти значения не вычисляются в сценарии).

Пример 2

Далее приведен код Web-страницы с формой, содержащей список с возможностью выбора сразу нескольких пунктов. При загрузке этой страницы особый сценарий делает пункты **JavaScript** и **Java** этого списка выбранными.

```
<HTML>
  <HEAD>
    <TITLE>Пример</TITLE>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Ваши любимые языки программирования:
      <SELECT ID="prefs" NAME="prefs" SIZE="4" MULTIPLE>
        <OPTION VALUE="javascript">JavaScript</OPTION>
        <OPTION VALUE="java">Java</OPTION>
        <OPTION VALUE="cpp">C++</OPTION>
        <OPTION VALUE="delphi">Borland Delphi</OPTION>
        <OPTION VALUE="vb">Microsoft Visual Basic</OPTION>
      </SELECT>
      <INPUT TYPE="submit" ID="btn" NAME="btn">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      var prefsObject = document.all["prefs"];
      var i = 0;
      var optionObject = null;
      for (i = 0; i < prefsObject.options.length; i++)
        {
          optionObject = prefsObject.options[i];
          if ((optionObject.value == "javascript") ||
          ↳(optionObject.value == "java"))
            optionObject.selected = true;
        }
    </SCRIPT>
  </BODY>
</HTML>
```

Как отследить момент изменения значения элемента управления?

Проблема

Мне нужно отследить момент, когда посетитель изменяет значение элемента управления (вводит новый текст в поле ввода, включает или отключает флажок и т. п.). Как это сделать?

Решение

Использовать обработчики соответствующих событий:

- `onChange` — для полей ввода и областей редактирования;
- `onClick` — для кнопок, флажков, переключателей и списков.

> ***Народ предупреждает!***
>
> В списках событие `onClick` возникает только при выборе пункта щелчком мыши. Выбор пункта списка с помощью клавиатуры не приводит к возникновению этого события. Так что полноценно отследить момент выбора пункта в списке не получится.
>
> Не забываем также, что событие `onChange` возникает только после того, как с данного поля ввода будет убран фокус.

Как временно сделать элемент управления недоступным?

Проблема

Мне нужно временно сделать элемент управления недоступным для ввода данных. Что для этого требуется?

Решение

Присвоить свойству `disabled` элемента управления значение `true`. После этого элемент управления станет недоступным для ввода данных, хоть и останется на странице. (Визуально такой элемент управления отображается несколько бледнее, чем обычный.) Чтобы снова сделать его доступным, нужно присвоить все тому же свойству `disabled` значение `false`.

Кроме того, поля ввода поддерживают свойство `readOnly`. Если этому свойству присвоить значение `true`, поле ввода останется доступным, но только для чтения; посетитель может выделить содержащийся в поле ввода текст и скопировать его в буфер обмена, но не сможет изменить или удалить его. Чтобы вернуть полю ввода возможность ввода текста, нужно присвоить его свойству `disabled` значение `false`.

И еще не нужно забывать о свойстве `visibility` внутреннего объекта `style`. Значение `"hidden"` этого свойства скрывает элемент управления, а значение `"visible"` снова делает его видимым.

Народ замечает

Даже если мы сделаем элемент управления недоступным или совсем скроем его, мы все еще сможем присвоить ему новое значение программно, из Web-сценария.

Пример

Далее приведен код HTML Web-страницы, содержащей форму для ввода данных о посетителе. При выборе переключателя **Опытный** или **Гуру** становится доступным поле ввода **Стаж**, позволяющее ввести стаж работы в годах.

```
<HTML>
  <HEAD>
    <TITLE>Страница с формой</TITLE>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Имя:
      <INPUT TYPE="text" ID="name" NAME="name"><BR><BR>
      E-mail:
      <INPUT TYPE="text" ID="email" NAME="email"><BR><BR>
      Квалификация:<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="novice"
      ↳ONCLICK="yearsObject.disabled = true;"> Новичок<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="advanced" CHECKED
      ↳ONCLICK="yearsObject.disabled = false;"> Опытный<BR>
      <INPUT TYPE="radio" ID="qual" NAME="qual" VALUE="guru"
      ↳ONCLICK="yearsObject.disabled = false;"> Гуру<BR><BR>
      Стаж:
      <INPUT TYPE="text" ID="years" NAME="years"><BR><BR>
      Подписаться на рассылку:
      <INPUT TYPE="checkbox" ID="maillist" NAME="maillist" CHECKED>
      ↳<BR><BR><BR>
      <INPUT TYPE="submit" ID="btn" NAME="btn">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      var frmObject = document.forms["frm"];
      var yearsObject = frmObject.elements["years"];
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь мы сделали включенным по умолчанию переключатель **Опытный**, чтобы однозначно определить состояние поля ввода **Стаж** — доступно оно для ввода данных или нет. Это обычная практика при создании пользовательского интерфейса программ, как обычных Windows-приложений, так и работающих на стороне сервера.

Народ замечает

Кстати, такие вот включающиеся и отключающиеся элементы управления очень часто применяются в Web-формах (да и в обычных Windows-приложениях). Если какие-либо элементы управления на форме в данный момент не нужны (в смысле, не нужны данные, которые в них вводятся), то их лучше временно сделать недоступными.

Как программно заполнить список?

Проблема

Я создаю сложную форму и хотел бы сделать список, набор пунктов которого менялся бы в зависимости от каких-то условий. Можно ли это сделать?

Решение

Конечно можно! Более того, для этого используются те же методы и свойства DOM, что мы изучили еще в *главе 2*.

Первое, что нам нужно сделать, — это подготовить полный набор пунктов, которые мы хотим видеть в списке. "Полный" — это значит включающий в себя все пункты, которые должны присутствовать в списке при любых условиях.

Далее нам нужно определить, какие данные характеризуют каждый пункт списка, включенный в этот набор. Это:

- название пункта (текст, выводимый на экране);
- значение (задается атрибутом `VALUE` тега `<OPTION>`; может отсутствовать);
- некий параметр, обозначающий группу пунктов списка, в которую входит данный пункт. Нужен для того, чтобы определить, в каком случае этот пункт должен присутствовать в списке. Может иметь строковое или числовое значение (второе зачастую предпочтительнее).

Мы можем расширить набор характеристик пункта, включив в него, например, признак того, должен ли данный пункт быть изначально выбранным в списке. Но это уже детали реализации.

Проще всего полный список пунктов организовать в виде массива. Каждый элемент такого массива представляет собой вложенный массив, три элемента

которого содержат три описанных выше характеристики пункта. (Можно также использовать три обычных одномерных массива, но это не очень удобно.)

Подготовив список пунктов, мы можем приняться за заполняющий список сценарий. Он будет всего один, но в его задачи будет входить:

- очистка списка от пунктов;
- определение пунктов, которые должны в данный момент входить в список;
- заполнение списка отобранными пунктами.

Очистить список от пунктов несложно. Мы проверяем, равно ли значение свойства `length` (количество элементов) коллекции `options` списка нулю, и, если не равно, удаляем первый элемент этой коллекции. Эти действия нужно повторять до тех пор, пока значение свойства `length` коллекции `options` не станет равным нулю, т. е. коллекция `options` опустеет (и список останется без всех пунктов).

Для очистки списка от пунктов можно применить функцию `jspsClearSelect` (листинг 14.7). Формат ее вызова приведен далее.

```
jspsClearSelect(<Список>);
```

Единственным параметром этой функции передается список, который нужно очистить. Функция `jspsClearSelect` не возвращает значения.

Листинг 14.7. Функция `jspsClearSelect`, выполняющая очистку списка от пунктов

```
function jspsClearSelect(pSelect)
  {
    while (pSelect.options.length > 0)
      pSelect.removeChild(pSelect.options[0]);
  }
```

Определение пунктов, которые в данный момент должны входить в список, следует выполнять по значению третьей характеристики каждого пункта (см. приведенный ранее список). Просто проверяем ее значение и на основе результата сравнения определяем, включать ли данный пункт в список или нет.

А для добавления пунктов в список следует пользоваться изученными в *главе 2* методами и свойствами DOM. Мы это проделывали не раз, так что и здесь трудности не возникнут.

Для добавления пункта в список можно также использовать функцию `jspsAddOptionToSelect` (листинг 14.8). Вот формат ее вызова:

```
jspsAddOptionToSelect(<Список>, <Название пункта>
↳ [, <Значение пункта>[, <Выбран изначально>]]);
```

Первым параметром этой функции передается список, в который нужно добавить новый пункт. Второй параметр должен содержать название добавляемого пункта в строковом виде. Третий, необязательный, параметр передает в функцию значение добавляемого пункта, которое также должно быть задано в строковом виде. Четвертый, также необязательный, параметр позволяет сделать пункт выбранным изначально; значение `true` делает пункт выбранным, а значение `false` или пропуск этого параметра — невыбранным.

Функция `jspsAddOptionToSelect` не возвращает значения.

Листинг 14.8. Функция `jspsAddOptionToSelect`, добавляющая новый пункт в список

```
function jspsAddOptionToSelect(pSelect, pText, pValue, pSelected)
  {
    if (typeof(pSelected) == "undefined")
      pSelected = false;
    var optionObject = document.createElement("OPTION");
    var textObject = document.createTextNode(pText);
    optionObject.appendChild(textObject);
    pSelect.appendChild(optionObject);
    if (pValue)
      optionObject.value = pValue;
    optionObject.selected = pSelected;
  }
```

Хорошая идея!

Добавьте объявление этих функций в файл сценариев controls.js. Впоследствии, чтобы использовать их, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="controls.js"></SCRIPT>
```

Пример

Далее приведен HTML-код Web-страницы с формой, позволяющей посетителю выбрать его любимые языки программирования. Языки программирования выбираются в списке **Языки**. При этом с помощью набора переключателей **Категория** можно указать категорию языков — интерпретируемые или компилируемые.

```
<HTML>
  <HEAD>
    <TITLE>Ваши любимые языки программирования</TITLE>
```

```
  <SCRIPT SRC="controls.js"></SCRIPT>
  <!-- Предполагается, что объявления функций для работы с элементами
  управления находятся в файле сценариев controls.js -->
  <SCRIPT TYPE="text/javascript">
    function catOnClick()
      {
        var frmObject = document.forms["frm"];
        var langsObject = frmObject.elements["langs"];
        jspsClearSelect(langsObject);
        var checkedRadioIndex = jspsGetCheckedRadioIndex(frmObject,
        ↳"cat");
        var i = 0;
        for (i = 0; i < optionsArray.length; i++)
          if (optionsArray[i][3] == checkedRadioIndex)
            jspsAddOptionToSelect(langsObject, optionsArray[i][0],
            ↳optionsArray[i][1], optionsArray[i][2]);
      }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM ID="frm" NAME="frm">
    Ваши любимые языки программирования<BR><BR>
    Категория:<BR>
    <INPUT TYPE="radio" ID="cat" NAME="cat" VALUE="compiling" CHECKED
    ↳ONCLICK="catOnClick();"> Компилируемые<BR>
    <INPUT TYPE="radio" ID="cat" NAME="cat" VALUE="interpreting"
    ↳ONCLICK="catOnClick();"> Интерпретируемые<BR>
    Языки:<BR>
    <SELECT ID="langs" NAME="langs" SIZE="4" MULTIPLE>
    </SELECT>
    <INPUT TYPE="submit" ID="btn" NAME="btn">
  </FORM>
  <SCRIPT TYPE="text/javascript">
    var optionsArray = new Array();
    optionsArray[0] = new Array("JavaScript", "javascript", true, 1);
    optionsArray[1] = new Array("Java", "java", true, 1);
    optionsArray[2] = new Array("Borland Delphi", "delphi", true, 0);
    optionsArray[3] = new Array("C++", "cpp", false, 0);
    optionsArray[4] = new Array("Perl", "perl", false, 1);
    optionsArray[5] = new Array("Microsoft Visual Basic", "vb", true,
    ↳0);
    optionsArray[6] = new Array("Microsoft Visual FoxPro", "foxpro",
    ↳false, 0);
```

```
      catOnClick();
    </SCRIPT>
  </BODY>
</HTML>
```

Внимание!

Приведенный пример использует функцию `jspsGetCheckedRadioIndex` (см. листинг 14.1).

В качестве полного списка пунктов мы использовали массив `optionsArray`, каждый элемент которого представляет собой вложенный массив. Элементы каждого такого вложенного массива содержат:

- название пункта;
- значение пункта;
- логический признак того, должен ли этот пункт быть изначально выбранным (значение `true`) или нет (значение `false`);
- номер включенного переключателя группы **Категория**. Переключатель **Компилируемые** имеет номер 0, а переключатель **Интерпретируемые** — номер 1. Мы используем числовые номера в качестве признака группы пунктов, т. к. с числами проще работать.

Народ советует

Действительно, числовые номера очень удобно использовать в качестве признака принадлежности чего-либо к какой-либо группе. Дело в том, что сравнение чисел (и логических величин) выполняется компьютером значительно быстрее, чем сравнение строк.

За заполнение списка пунктами "отвечает" обработчик события `onClick` обоих переключателей группы **Категория**, носящий имя `catOnClick`. Отметим, что изначально первый переключатель (**Компилируемые**) этой группы включен — мы вставили в создающий его тег `<OPTION>` атрибут `CHECKED`. А в самом конце кода Web-страницы вызвали функцию `catOnClick`, чтобы сразу заполнить список компилируемыми языками программирования.

Народ советует

В группе должен быть изначально включен хотя бы один переключатель. Группа, состоящая из одних отключенных переключателей, — плохой тон программирования.

Более сложные манипуляции с формами и элементами управления

От простого — к сложному. Сейчас мы выясним, как ограничить ввод посетителя в поле ввода только определенным набором символов (например,

только числами) и как выполнить проверку введенных данных на корректность.

Как ограничить набор символов, вводимых посетителем в поле ввода?

Проблема

Я создал форму с полем ввода, в который должно вводиться число. Как мне запретить ввод в это поле всех символов, отличных от чисел?

Решение (Internet Explorer)

Создать обработчик события `onKeyPress` и проверять в нем значение свойства `keyCode` объекта `event`. (Как мы помним, в этом свойстве находится код введенного символа в формате Unicode.) Если посетитель ввел недопустимый символ (скажем, букву), присваиваем этому свойству значение 0, чтобы подавить ввод этого символа.

Чтобы получить код нужного нам символа в формате Unicode, достаточно воспользоваться приложением Таблица символов, стандартно поставляемым в составе Windows 2000/XP/2003 Server. Запустить его можно, щелкнув ярлык **Таблица символов** в меню **Пуск | Стандартные программы | Служебные**. Так вот, если выделить щелчком мыши любой символ в окне этого приложения, то в строке статуса его окна появится нужный нам код символа. Нам останется только удалить из него начальные символы "U+" и подставить в сценарий.

Народ предупреждает!

Ох, опять морока с несовместимостью Web-обозревателей!.. Из-за нее этот способ работает только в Internet Explorer. Opera не позволяет менять значение свойства `keyCode` объекта `event`. Что касается Firefox, то он не позволяет получить код введенного символа в обработчике события.

Народ советует

Так что таким способом обеспечить правильность ввода данных лучше не пользоваться.

Пример

Далее приведен HTML-код Web-страницы с формой, которая содержит поле ввода **Количество**. В это поле ввода возможно ввести только цифры от 0 до 9.

```
<HTML>
  <HEAD>
```

```
    <TITLE>Только числа</TITLE>
    <SCRIPT TYPE="text/javascript">
      function countOnKeyPress()
        {
          if (!((event.keyCode >= 0x30) && (event.keyCode <= 0x39)))
            event.keyCode = 0;
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Количество:
      <INPUT TYPE="text" ID="count" NAME="count"
      ↳ONKEYPRESS="countOnKeyPress();"><BR>
      <INPUT TYPE="submit" ID="btn" NAME="btn">
    </FORM>
  </BODY>
</HTML>
```

В функции-обработчике события `onKeyPress` поля ввода `count` мы проверяем, находится ли код введенного посетителем символа между `0x30` (шестнадцатеричный код символа "0") и `0x39` (шестнадцатеричный код символа "9"). Если же это не так, мы подавляем ввод недопустимого символа присвоением свойству `keyPress` объекта `event` значения 0.

> ***Народ сокрушается***
>
> Неплохой, кстати, способ обеспечить корректный ввод данных, так сказать, "принудительно". Жаль, что работает только в Internet Explorer...

Как проверить введенные посетителем данные на корректность?

Проблема

Мне нужно проверить введенные в форму данные на корректность до их отправки серверной программе. Как это проще всего сделать?

Решение 1

Проще всего это сделать в обработчике события `onSubmit` формы (это событие, как мы помним, возникает в форме перед отправкой данных серверной программе). Если введенные посетителем данные некорректны, обработчик должен прервать отправку данных формой, вернув значение `false`.

Народ замечает

Думается, проверять на корректность стоит только содержимое полей ввода и областей редактирования. Эти элементы управления допускают значительно больше свободы при вводе данных, чем какие-нибудь флажки или списки, поэтому и вероятность допустить ошибку при вводе в них данных весьма велика.

Пример

Далее приведен HTML-код Web-страницы с формой. Эта форма содержит поле ввода числового значения. Если в него введено значение, не являющееся числовым, на экране появится окно-сообщение с предупреждающим текстом, а отправка данных выполнена не будет.

```
<HTML>
  <HEAD>
    <TITLE>Только числа!</TITLE>
    <SCRIPT TYPE="text/javascript">
      function checkFormData()
        {
          var frmObject = document.forms["frm"];
          var countObject = frmObject.elements["count"];
          if (isNaN(countObject.value))
            {
              window.alert("Введите, пожалуйста, число");
              return false;
            }
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm" ONSUBMIT="checkFormData();">
      Количество:
      <INPUT TYPE="text" ID="count" NAME="count"><BR>
      <INPUT TYPE="submit" ID="btn" NAME="btn">
    </FORM>
  </BODY>
</HTML>
```

Здесь для проверки введенного в поле ввода на то, является ли оно числом, мы использовали стандартную функцию JavaScript `isNaN`. Эта функция принимает в качестве единственного параметра строку и возвращает `true`, если она *не* может быть преобразована в число. В противном случае, т. е. если строка может быть преобразована в число, возвращается `false`.

Решение 2

Использовать универсальную функцию `jspsCheckFormData` (листинг 14.9), выполняющую проверку данных, содержащихся в полях ввода и областях редактирования (значения остальных элементов управления она не проверяет). Формат ее вызова таков:

```
jspsCheckFormData(<Имя формы>, <Массив проверок>);
```

Первым параметром этой функции передается имя формы, данные которой нужно проверить. Это имя передается в строковом виде.

Вторым параметром функции `jspsCheckFormData` передается так называемый массив проверок. Каждый элемент этого массива содержит описание проверки, которую нужно выполнить над содержимым какого-либо поля ввода, и представляет собой вложенный массив, содержащий три элемента:

- имя поля ввода, содержимое которого нужно проверить, в строковом виде;
- функция, выполняющая проверку данных в этом поле ввода (о ней мы подробно поговорим чуть позже);
- сообщение, выводимое на экран, если проверка будет неудачной. Задается в строковом виде. Если задана пустая строка, никакого сообщения не выводится (в этом случае вывод сообщения нужно предусмотреть в самой функции, выполняющей проверку).

Все эти проверки выполняются в том порядке, в каком они внесены в массив.

Если все проверки выполнились успешно, функция `jspsCheckFormData` возвращает значение `true`. Если же одна из проверок прошла неудачно, остальные проверки не выполняются, функция завершает свою работу и возвращает значение `false`.

> ***Народ советует***
>
> Возвращаемый функцией `jspsCheckFormData` результат очень удобно использовать для прерывания отправки данных в случае неуспешности проверок. Для этого достаточно создать такой обработчик события `onSubmit` формы:
>
> ```
> return jspsCheckFormData(<Параметры>);
> ```

Что касается функции, выполняющей проверку, то она должна иметь такой формат вызова:

```
<Функция проверки>(<Форма>, <Поле ввода или область редактирования>,
↳<Значение элемента управления>);
```

Первый параметр, который должна принимать *Функция проверки*, есть форма, содержащая элемент управления (поле ввода или область редактирования), значение которого нужно проверить на корректность. Сам этот элемент

управления передается вторым параметром. А третьим параметром передается значение этого элемента управления, которое, собственно, и нужно проверить, в строковом виде.

Функция проверки должна возвращать `true`, если проверка прошла удачно, и `false` в противном случае.

Для проверки данных можно использовать три "стандартные" функции, объявленные в листинге 14.9 вместе с функцией `jspsCheckFormData`. Эти функции доступны через следующие псевдоконстанты:

- `JSPS_CFD_CF_NON_EMPTY` — поле ввода должно содержать любое значение;
- `JSPS_CFD_CF_IS_NUMBER` — поле ввода должно содержать числовое значение;
- `JSPS_CFD_CF_IS_EMAIL` — поле ввода должно содержать адрес электронной почты.

Листинг 14.9. Функция `jspsCheckFormData`, выполняющая проверку введенных в форму данных на корректность

```
// Объявляем описанные выше функции проверки: на непустое значение,
// на число и на адрес электронной почты
function jspsCFDCFNonEmpty(pForm, pElement, pValue)
  {
    return (pValue != "");
  }

function jspsCFDCFIsNumber(pForm, pElement, pValue)
  {
    return isNaN(pValue);
  }

function jspsCFDCFIsEmail(pForm, pElement, pValue)
  {
    return (pValue.indexOf("@") > -1);
  }

// Объявляем псевдоконстанты для вызова объявленных выше функций проверки
JSPS_CFD_CF_NON_EMPTY = jspsCFDCFNonEmpty;
JSPS_CFD_CF_IS_NUMBER = jspsCFDCFIsNumber;
JSPS_CFD_CF_IS_EMAIL = jspsCFDCFIsEmail;

// Объявляем саму функцию jspsCheckFormData
function jspsCheckFormData(pFormName, pCheckArray)
```

```
{
  var frmObject = document.forms[pFormName];
  var checkCount = pCheckArray.length;
  var i = 0;
  var returnValue = true;
  for (i = 0; i < checkCount; i++)
    {
      var elementObject = frmObject.elements[pCheckArray[i][0]];
      returnValue = pCheckArray[i][1](frmObject, elementObject,
      ↳elementObject.value);
      if (!(returnValue))
        {
          if (pCheckArray[i][2]) window.alert(pCheckArray[i][2]);
          break;
        }
    }
  return returnValue;
}
```

Хорошая идея!

Добавьте объявление этой функции в файл сценариев controls.js. Впоследствии, чтобы использовать ее, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="controls.js"></SCRIPT>
```

Пример

Далее приведен HTML-код Web-страницы с формой регистрации нового посетителя сайта. Эта форма содержит поля ввода для задания адреса электронной почты в качестве имени посетителя и пароля. С помощью объявленной в листинге 14.9 функции `jspsCheckFormData` выполняется проверка на корректность ввода адреса электронной почты и достаточную "стойкость" пароля (он не должен быть короче 8 символов).

```
<HTML>
  <HEAD>
    <TITLE>Регистрация нового посетителя</TITLE>
    <SCRIPT SRC="controls.js"></SCRIPT>
    <!-- Предполагается, что объявление функции jspsCheckFormData
    находится в файле сценариев controls.js -->
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
```

```
      // Объявляем функцию, выполняющую проверку "стойкости" введенного
      // пароля (его длина должна быть не менее 8 символов)
      function isCorrectPassword(pForm, pElement, pValue)
        {
          return (pValue.length >= 8);
        }

      var checkArray = new Array();
      checkArray[0] = new Array("email", JSPS_CFD_CF_NON_EMPTY,
      ↳"Введите, пожалуйста, свой адрес e-mail");
      checkArray[1] = new Array("email", JSPS_CFD_CF_IS_EMAIL,
      ↳"Введите, пожалуйста, правильный адрес e-mail");
      checkArray[2] = new Array("pass", JSPS_CFD_CF_NON_EMPTY,
      ↳"Введите, пожалуйста, пароль");
      checkArray[3] = new Array("pass", isCorrectPassword,
      ↳"Пароль не должен быть короче 8 символов");
    </SCRIPT>
    <FORM ID="frm" NAME="frm"
    ↳ONSUBMIT="return jspsCheckFormData('frm', checkArray);">
      E-mail:
      <INPUT TYPE="text" ID="email" NAME="email"><BR>
      Пароль:
      <INPUT TYPE="password" ID="pass" NAME="pass"><BR>
      <INPUT TYPE="submit" ID="btn" NAME="btn">
    </FORM>
  </BODY>
</HTML>
```

Заметим, что мы сначала выполняем проверку на то, что в оба поля ввода вообще что-то введено, для чего используем "стандартную" функцию `JSPS_CFD_CF_NON_EMPTY`. Далее мы проверяем содержимое поля ввода адреса электронной почты на корректность самого введенного в него адреса, применяя другую "стандартную" функцию — `JSPS_CFD_CF_IS_EMAIL`. А для проверки достаточной "стойкости" пароля мы используем написанную собственноручно функцию `isCorrectPassword`.

Как использовать диалоговые окна HTML?

Проблема

Я что-то слышал о диалоговых окнах HTML, якобы поддерживаемых Internet Explorer. Что это такое и как их использовать?

Решение

Диалоговые окна HTML — это обычные окна Web-обозревателя, открываемые программно (из Web-сценария) и содержащие какую-либо Web-страницу, как правило, с Web-формой. Их отличие от обычных окон Web-обозревателя в том, что они всегда перекрывают обычные окна, даже если в данный момент неактивны.

Диалоговые окна HTML бывают двух видов. *Модальное* диалоговое окно не только находится поверх создавшего его окна Web-обозревателя, но и не позволяет посетителю переключиться на него. *Немодальное* же диалоговое окно такую возможность предоставляет.

> ***Народ предупреждает!***
>
> Диалоговые окна HTML поддерживаются только Internet Explorer, начиная с версии 4.0.

Модальные диалоговые окна HTML обычно используют для того, чтобы получить от посетителя какие-либо данные, необходимые для дальнейшей работы сценария (например, имя и пароль для входа в закрытый раздел сайта). Пока посетитель не введет необходимые данные в модальное окно и не нажмет кнопку **OK** или **Отмена**, он не сможет продолжить работу с сайтом или Web-приложением. Область применения немодальных диалоговых окон заметно у́же — это ввод различных вспомогательных данных или задание параметров, влияющих на отображение информации на Web-странице.

Давайте же рассмотрим работу с диалоговыми окнами HTML, как модальными, так и немодельными. И начнем как раз с модальных окон как наиболее часто используемых.

Для вывода на экран модального диалогового окна HTML используется метод `showModalDialog` объекта `window`. Вот формат его вызова:

```
window.showModalDialog(<Интернет-адрес Web-страницы>
↳[, <Данные, передаваемые диалоговому окну>
↳[, <Параметры диалогового окна>]]);
```

Первый параметр этого метода задает интернет-адрес Web-страницы, которая станет содержимым диалогового окна HTML. Понятно, что он должен быть задан в строковом виде. Поскольку диалоговые окна обычно используются для ввода данных, эта страница должна содержать Web-форму с соответствующими элементами управления.

Второй параметр метода `showModalDialog` объекта `window` задает данные, которые должны быть переданы диалоговому окну HTML (точнее, открытой в нем Web-странице). Это может быть как одна-единственная переменная, так и целый массив или экземпляр объекта, если передаваемых данных слишком

много. Как диалоговое окно получает эти данные и как потом возвращает вызвавшему его сценарию результат, мы рассмотрим потом.

Третий параметр задает строку, содержащую список параметров создаваемого диалогового окна, разделенных запятыми. Здесь тот же принцип, что и у третьего параметра метода `open` объекта `window`, рассмотренного нами в *главе 4*. Все доступные для задания параметры перечислены в табл. 14.5.

Таблица 14.5. *Параметры диалогового окна HTML*

Свойство окна	Описание
`center=yes|no`	Если `yes`, то создаваемое окно будет находиться в центре экрана. Значение по умолчанию — `yes`
`dialogHeight:<`*Значение*`>`	Высота создаваемого окна
`dialogHide=yes|no`	Если `yes`, то создаваемое окно будет скрываться при печати или предварительном просмотре перед печатью. Значение по умолчанию — `no`
`dialogLeft:<`*Значение*`>`	Горизонтальная координата левого верхнего угла создаваемого окна
`dialogTop:<`*Значение*`>`	Вертикальная координата левого верхнего угла создаваемого окна
`dialogWidth:<`*Значение*`>`	Ширина создаваемого окна
`edge=sunken|raised`	Задает вид границы окна: вдавленный (`sunken`) или выпуклый (`raised`). Значение по умолчанию — `raised`
`help=yes|no`	Если `yes`, то создаваемое окно будет отображать в строке заголовка кнопку вызова контекстной справки. Значение по умолчанию — `yes`
`resizable=yes|no`	Если `yes`, то посетитель сможет изменять размеры создаваемого окна. Значение по умолчанию — `no`
`scroll=yes|no`	Если `yes`, то создаваемое окно будет отображать полосы прокрутки, если содержимое Web-страницы не помещается в нем. Значение по умолчанию — `yes`
`status=yes|no`	Если `yes`, то создаваемое окно будет содержать строку статуса. Значение по умолчанию может в разных случаях быть различным, поэтому лучше задать это свойство явно
`unadorned=yes|no`	Если `yes`, то создаваемое окно будет "украшено". Что это значит, автору установить не удалось, т. к. никакой видимой разницы между "украшенным" и "неукрашенным" окном нет. Значение по умолчанию — `no`

Осталось сказать, что при задании местоположения (параметры `dialogLeft` и `dialogTop`) и размеров (`dialogWidth` и `dialogHeight`) диалогового окна нужно

также явно указать единицу измерения. Internet Explorer поддерживает все единицы измерения, содержащиеся в стандарте CSS, например, пикселы (обозначается как `px`) или миллиметры (`mm`).

Метод `showModalDialog` объекта `window` сразу после вызова выводит на экран диалоговое окно с заданной страницей и заданными параметрами. При этом выполнение сценария, содержащего вызов этого метода, приостанавливается, пока диалоговое окно не будет закрыто. После закрытия диалогового окна метод `showModalDialog` объекта `window` в качестве результата возвращает данные, переданные от диалогового окна (точнее, от открытой в нем страницы).

Давайте рассмотрим вот такое выражение:

```
var result = window.showModalDialog("form.html", [12, 45],
"resizable=no,scroll=no,status=no");
```

Оно выводит на экран модальное диалоговое окно HTML, содержащее страницу form.html, с неизменными размерами, без полос прокрутки и строки статуса. Кроме того, оно передает созданному диалоговому окну массив с числами 12 и 45. После закрытия диалогового окна этот метод помещает в переменную `result` данные, переданные диалоговым окном, и выполнение сценария продолжается.

> ***Народ советует***
>
> Практически все диалоговые окна имеют неизменяемые размеры, не содержат полос прокрутки и строки статуса. Модальные диалоговые окна, кроме того, всегда выводятся в центре экрана. Так что приведенная выше строка параметров диалогового окна является, можно сказать, стандартом. Правда, в ней не хватает размеров диалогового окна, но их можно подобрать только экспериментально.

Итак, модальное диалоговое окно открыто. Обратимся к открытой в ней странице (ее интернет-адрес, как мы уже знаем, передается первым параметром метода `showModalDialog` объекта `window`). Как она может получить отправленные через второй параметр этого метода данные? Очень просто — через свойство `dialogArguments` того же объекта `window`. Вот так:

```
var parArray = window.dialogArguments;
```

И в переменной `parArray` окажется переданный вторым параметром метода `showModalDialog` массив с двумя числами 12 и 45.

Можно написать и вот такой код:

```
var par1 = window.dialogArguments[0];
var par2 = window.dialogArguments[1];
```

т. е. сразу же извлечь числа, благо свойство `dialogArguments` и так содержит массив. Тогда в переменной `par1` окажется число 12, а в переменной `par2` — число 45.

Теперь мы можем делать с полученными данными все, что пожелаем. Например, мы можем подставить эти данные в элементы управления Web-формы страницы, открытой в диалоговом окне, в качестве изначальных значений. Так, кстати, часто делают.

Хорошо! Посетитель ввел данные в форму и нажал кнопку **OK**. Теперь диалоговое окно нужно закрыть и передать введенные в него данные окну, которое его открыло. Для этого случая объект `window` поддерживает свойство `returnValue`. Присваиваем этому свойству данные, которые нужно вернуть в открывшую это окно страницу, например:

```
window.returnValue = 123;
```

или

```
window.returnValue = ["user", "password"];
```

после чего закрываем диалоговое окно:

```
window.close();
```

Метод `showModalDialog` в сценарии, создавшем диалоговое окно, возвращает переданные из диалогового окна данные. И мы можем их использовать.

> ***Народ замечает***
>
> Кроме того, объект `window` поддерживает свойства `dialogLeft`, `dialogTop`, `dialogWidth` и `dialogHeight`. Эти свойства соответствуют описанным в табл. 14.5 параметрам диалогового окна.

Правила хорошего тона программирования рекомендуют оснащать диалоговые окна также и кнопкой **Отмена**. При нажатии этой кнопки диалоговое окно будет просто закрываться вызовом метода `close` объекта `window` без передачи каких-либо данных. Тогда метод `showModalDialog` вернет значение `null`.

Так, с модальными диалоговыми окнами HTML мы разобрались. Перейдем к немодальным.

Для вывода немодального диалогового окна HTML на экран используется другой метод объекта `window` — `showModelessDialog`. Вот формат его вызова:

```
window.showModelessDialog(<Интернет-адрес Web-страницы>
↳[, <Данные, передаваемые диалоговому окну>
↳[, <Параметры диалогового окна>]]);
```

Как видим, параметры этого метода такие же, как и у его "коллеги" `showModalDialog`. Возвращает метод `showModelessDialog` экземпляр объекта, соответствующий открытому диалоговому окну.

Сразу после вызова метода `showModelessDialog` объекта `window` немодальное диалоговое окно создается и выводится на экран. При этом сценарий, содержащий вызов этого метода, продолжает выполняться.

Стоп, но как же нам передать данные из немодального диалогового окна странице, что его открыла? Стандартного способа сделать это нет... Придется изобретать очередной "финт ушами"...

Сначала в странице, которая выводит на экран немодальное диалоговое окно, нам будет нужно объявить набор переменных для обмена данными — по одной переменной на каждую единицу данных. Перед выводом диалогового окна мы присваиваем этим переменным все необходимые значения, например:

```
var param1 = 100;
var param2 = "";
```

После этого можно вывести на экран немодальное диалоговое окно, воспользовавшись таким синтаксисом:

```
window.showModelessDialog(<Интернет-адрес Web-страницы>, window,
↳[, <Параметры диалогового окна>]]);
```

Заметим, что вторым параметром метода `showModelessDialog` мы передаем текущее окно Web-обозревателя.

В странице, открытой в диалоговом окне, получаем переданное окно:

```
var parentWindow = window.dialogArguments;
```

после чего получаем сами переданные данные, извлекая их из объявленных ранее переменных:

```
var tempParam1 = parentWindow.param1;
var tempParam2 = parentWindow.param2;
```

Если же нам нужно передать какие-либо данные из диалогового окна создавшей его странице, мы просто присваиваем их объявленным ранее переменным:

```
parentWindow.param1 = 200;
parentWindow.param2 = "superuser";
```

Иногда нам нужно при пересылке данных из диалогового окна создавшей его странице выполнять еще какие-то действия. Для этого мы можем объявить в странице, создающей окно, особую функцию:

```
function applyData()
  {
    // Здесь мы что-либо делаем с принятыми от диалогового окна данными
  }
```

После передачи данных мы вызываем эту функцию из немодального диалогового окна:

```
parentWindow.param1 = 200;
parentWindow.param2 = "superuser";
parentWindow.applyData();
```

Так, кстати, часто и делают.

Обычно немодальное диалоговое окно отправляет данные при нажатии кнопки **OK** или **Применить**. Также правила хорошего тона программирования рекомендуют оснащать немодальные окна кнопкой **Закрыть**, которая при нажатии закрывает это окно.

Пример 1

Далее приведен HTML-код двух Web-страниц, из которых одна является содержимым модального диалогового окна, позволяющего выбрать цвет текста другой страницы (основной) и сделать его полужирным.

Вот код основной страницы, которая вызывает диалоговое окно:

```
<HTML>
  <HEAD>
    <TITLE>Страница, вызывающая диалоговое окно</TITLE>
    <SCRIPT TYPE="text/javascript">
      // Объявляем переменные, хранящие текущие цвет и признак
      // "полужирности" текста
      var textColor = "#000000";
      var isTextBold = false;

      function btnOnClick()
        {
          // Выводим диалоговое окно и передаем ему текущие значения
          // параметров текста
          var result = window.showModalDialog("jspsModalDialog.html",
          ↳[textColor, isTextBold], "resizable=no,scroll=no,status=no");
          // Проверяем, действительно ли диалоговое окно передало
          // какие-то данные (т. е. нажал ли посетитель кнопку OK
          // этого окна)
          if (result)
            {
              // Если данные были переданы, обновляем параметры текста
              textColor = result[0];
              isTextBold = result[1];
              var parObject = document.all["par"];
              parObject.style.color = textColor;
              parObject.style.fontWeight = isTextBold ? "bold" :
              ↳"normal";
            }
        }
```

```
    </SCRIPT>
  </HEAD>
  <BODY>
    <P ID="par">Цвет этого текста можно изменить, воспользовавшись
    ↳модальным диалоговым окном.</P>
    <FORM ID="frm" NAME="frm">
      <INPUT TYPE="button" ID="btn" NAME="btn"
      ↳VALUE="Задать параметры текста" ONCLICK="btnOnClick();">
    </FORM>
  </BODY>
</HTML>
```

Сохраним этот код в файле jspsModalDialogCaller.html.

А вот HTML-код страницы, которая станет содержимым диалогового окна:

```
<HTML>
  <HEAD>
    <TITLE>Диалоговое окно</TITLE>
    <SCRIPT SRC="controls.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с элементами
    управления находятся в файле сценариев controls.js -->
    <SCRIPT TYPE="text/javascript">
      // При нажатии кнопки OK передаем данные основному окну
      function btnOKOnClick()
        {
          window.returnValue = [jspsGetCheckedRadioValue(frmObject,
          ↳"color"), boldObject.checked];
          window.close();
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Цвет текста:<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#000000">
      Черный<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#FF0000">
      Красный<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#00FF00">
      Зеленый<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#0000FF">
      Синий<BR>
```

```
      Полужирный текст:
      <INPUT TYPE="checkbox" ID="bold" NAME="bold"><BR><BR><BR>
      <INPUT TYPE="button" ID="btnOK" NAME="btnOK" VALUE="OK"
      ↳ONCLICK="btnOKOnClick();"><BR><BR>
      <INPUT TYPE="button" ID="btnCancel" NAME="btnCancel" VALUE="Отмена"
      ↳ONCLICK="window.close();">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      // Заполняем форму текущими значениями параметров текста,
      // переданными из основного окна
      var frmObject = document.forms["frm"];
      var boldObject = frmObject.elements["bold"];
      var colorObject = jspsGetRadioByValue(frmObject, "color",
      ↳window.dialogArguments[0]);
      colorObject.checked = true;
      boldObject.checked = window.dialogArguments[1];
    </SCRIPT>
  </BODY>
</HTML>
```

Внимание!

Приведенный пример использует функции `jspsGetCheckedRadioValue` (см. листинг 14.2) и `jspsGetRadioByValue` (см. листинг 14.6).

А этот код мы сохраним в файле jspsModalDialog.html.

Откроем страницу jspsModalDialogCaller.html в Internet Explorer и нажмем кнопку **Задать параметры текста**. Когда на экране появится диалоговое окно HTML, зададим с его помощью какие-либо параметры и нажмем кнопку **OK** этого окна. Цвет и "полужирность" текста основной страницы должны измениться.

Пример 2

Изменим страницы из первого примера таким образом, чтобы для задания параметров текста основной страницы использовалось немодальное диалоговое окно.

HTML-код основной страницы будет выглядеть так:

```
<HTML>
  <HEAD>
    <TITLE>Страница, вызывающая диалоговое окно</TITLE>
    <SCRIPT TYPE="text/javascript">
      // Объявляем переменные, хранящие текущие цвет и признак
      // "полужирности" текста
```

```
      var textColor = "#000000";
      var isTextBold = false;

      function btnOnClick()
        {
          // Выводим диалоговое окно и передаем ему текущее окно
          window.showModelessDialog("jspsNonModalDialog.html", window,
          ↳"resizable=no,scroll=no,status=no");
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <P ID="par">Цвет этого текста можно изменить, воспользовавшись
    ↳немодальным диалоговым окном.</P>
    <FORM ID="frm" NAME="frm">
      <INPUT TYPE="button" ID="btn" NAME="btn"
      ↳VALUE="Задать параметры текста" ONCLICK="btnOnClick();">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      // Объявляем переменную, содержашую абзац, параметры текста
      // которого мы будем задавать в диалоговом окне
      var parObject = document.all["par"];
    </SCRIPT>
  </BODY>
</HTML>
```

Сохраним этот код в файле jspsNonModalDialogCaller.html.

А вот исправленный HTML-код страницы, которая станет содержимым диалогового окна:

```
<HTML>
  <HEAD>
    <TITLE>Диалоговое окно</TITLE>
    <SCRIPT SRC="controls.js"></SCRIPT>
    <!-- Предполагается, что объявления функций для работы с элементами
    управления находятся в файле сценариев controls.js -->
    <SCRIPT TYPE="text/javascript">
      // При нажатии кнопки "Применить" применяем заданные параметры к
      // тексту абзаца основной страницы. Обратим внимание, что при этом
      // также обновляем текущие параметры, хранящиеся в переменных
      // textColor и isTextBold основной страницы
      function btnApplyOnClick()
        {
          callerObject.textColor = jspsGetCheckedRadioValue(frmObject,
          ↳"color");
```

```
          callerObject.parObject.style.color = callerObject.textColor;
          callerObject.isTextBold = boldObject.checked;
          callerObject.parObject.style.fontWeight = boldObject.checked ?
          ↳"bold" : "normal";
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM ID="frm" NAME="frm">
      Цвет текста:<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#000000">
      Черный<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#FF0000">
      Красный<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#00FF00">
      Зеленый<BR>
      <INPUT TYPE="radio" ID="color" NAME="color" VALUE="#0000FF">
      Синий<BR>
      Полужирный текст:
      <INPUT TYPE="checkbox" ID="bold" NAME="bold"><BR><BR><BR>
      <INPUT TYPE="button" ID="btnApply" NAME="btnApply"
      ↳VALUE="Применить" ONCLICK="btnApplyOnClick();"><BR><BR>
      <INPUT TYPE="button" ID="btnClose" NAME="btnClose" VALUE="Закрыть"
      ↳ONCLICK="window.close();">
    </FORM>
    <SCRIPT TYPE="text/javascript">
      // Заполняем форму текущими значениями параметров текста
      var callerObject = window.dialogArguments;
      var frmObject = document.forms["frm"];
      var boldObject = frmObject.elements["bold"];
      var colorObject = jspsGetRadioByValue(frmObject, "color",
      ↳callerObject.textColor);
      colorObject.checked = true;
      boldObject.checked = callerObject.isTextBold;
    </SCRIPT>
  </BODY>
</HTML>
```

Внимание!

Приведенный пример использует функции `jspsGetCheckedRadioValue` (см. листинг 14.2) и `jspsGetRadioByValue` (см. листинг 14.6).

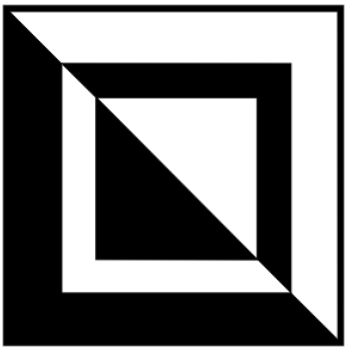
А этот код мы сохраним в файле jspsNonModalDialog.html.

Откроем страницу jspsNonModalDialogCaller.html в Internet Explorer и нажмем кнопку **Задать параметры текста**. Когда на экране появится диалоговое окно HTML, зададим с его помощью какие-либо параметры и нажмем кнопку **Применить** этого окна. Текст основной страницы должен измениться.

Что дальше?

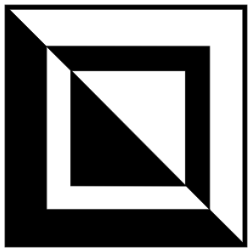
Ну вот мы и разобрались с вводом, выводом, сохранением и передачей данных. Разговор на эту тему был долгим, но плодотворным. Народ — лучший учитель!

Время идет, и книга подходит к концу. Нам осталось только немного попрактиковаться в сложном Web-программировании, изучить еще некоторые возможности, предлагаемые программистам Internet Explorer, и поговорить об отладке Web-сценариев. Все это займет две последние главы.



ЧАСТЬ V

Сложное Web-программирование



ГЛАВА 15

Приемы сложного Web-программирования

Напоследок мы займемся сложным Web-программированием, с привлечением всего того, что изучили в предыдущих главах этой книги. Мы научимся создавать страницы, состоящие из множества вкладок, и реализовывать слайд-шоу, а под конец мы изучим еще две возможности неисчерпаемого Internet Explorer: работу с внешними текстовыми базами данных и создание HTML-приложений. Если мы собираемся в будущем заниматься разработкой корпоративных решений на основе Internet Explorer, все это нам пригодится.

Как создать страницу, состоящую из нескольких вкладок?

Проблема

Многие окна Windows-приложений имеют несколько вкладок с разным содержимым. Можно ли сделать то же самое на Web-странице?

Решение

Можно. Более того, никакого особо сложного программирования нам для этого не понадобится.

1. Создаем набор гиперссылок, предназначенных для переключения между вкладками. Также создаем стили CSS для этих гиперссылок, чтобы сделать их более похожими на корешки вкладок Windows.
2. Создаем общий контейнер `<DIV>`, вмещающий все контейнеры, которые и станут вкладками. Также создаем для этого контейнера стиль CSS, чтобы сделать его более похожим на стандартный набор вкладок Windows.
3. Внутри этого контейнера создаем набор контейнеров `<DIV>`, которые станут вкладками. Делаем все эти контейнеры скрытыми (для чего достаточ-

но создать стиль CSS, содержащий атрибут `display` со значением "`none`", и привязать его к этим контейнерам), за исключением контейнера, соответствующего изначально видимой вкладке.

4. Пишем функцию-обработчик события `onClick` гиперссылок, созданных на шаге 1. Эта функция будет делать нужный нам контейнер-вкладку видимым (для этого следует присвоить свойству `display` объекта `style` значение "`block`"), а все остальные — невидимыми (значение "`none`" свойства `display` объекта `style`). Также она будет менять вид гиперссылок-корешков таким образом, чтобы выделить гиперссылку, соответствующую видимой в данный момент вкладке. Номер контейнера-вкладки, который нужно сделать видимым, можно передавать через параметр этой функции-обработчика.
5. Привязываем созданную на шаге 4 функцию-обработчик к событиям `onClick` гиперссылок, созданных на шаге 1. (Также можно поместить вызов этой функции в атрибут `HREF` тега `<A>`, предварив его символами "javascript:".)

Вместо гиперссылок для переключения между вкладками также можно использовать кнопки. Правда, выглядеть это будет не очень изящно.

Народ советует

Часто "многовкладочными" делают Web-формы, что позволяет поместить их на ограниченном пространстве. Но иногда и содержимое обычной Web-страницы "разносят" по нескольким вкладкам.

Народ предупреждает!

Если создается "многовкладочная" Web-форма, код, создающий и гиперссылки-корешки, и сами контейнеры-вкладки, следует помещать в теге `<FORM>`. Иначе Web-обозреватель посчитает содержимое разных вкладок отдельными формами. Также нужно иметь в виду, что кнопка **Отправить** (и кнопка **Сброс**, если она присутствует в форме) должна находиться вне вкладок.

Пример

Далее приведен HTML-код "многовкладочной" Web-страницы, каждая из трех вкладок которой содержит свое собственное содержимое.

```
<HTML>
  <HEAD>
    <TITLE>Страница с вкладками</TITLE>
    <SCRIPT TYPE="text/javascript">
      // Объявляем функцию, выполняющую переключение между вкладками
      function showTab(pTabIndex)
        {
          var i = 0;
```

```
        for (i = 0; i < tabs.length; i++)
          {
            // Делаем нужный контейнер-вкладку видимой,
            // остальные — невидимыми
            tabs[i].style.display = (i == pTabIndex) ? "block" :
            ↳"none";
            // Выделяем гиперссылку-корешок, соответствующую видимой
            // вкладке, серым фоном, остальные — белым
            tabHs[i].style.backgroundColor = (i == pTabIndex) ?
            ↳"#CCCCCC" : "#FFFFFF";
          }
      }
  </SCRIPT>
  <!-- Создаем стилевые классы для контейнера, содержащего
  гиперссылки-корешки вкладок, и для контейнера, содержащего сами
  вкладки -->
  <STYLE>
    .tabhost   { border: thin solid #CCCCCC;
                 padding: 5px; }
    .tabhs A   { border: thin solid #CCCCCC;
                 padding: 5px 5px 0px 5px; }
  </STYLE>
</HEAD>
<BODY>
  <!-- Создаем гиперссылки-корешки и помещаем их в контейнер.
  Так нам будет проще применить к ним созданный ранее стилевой
  класс -->
  <DIV CLASS="tabhs">
    <A ID="tabH1" HREF="#" ONCLICK="showTab(0);">Вкладка 1</A> 
    <A ID="tabH2" HREF="#" ONCLICK="showTab(1);">Вкладка 2</A> 
    <A ID="tabH3" HREF="#" ONCLICK="showTab(2);">Вкладка 3</A>
  </DIV>
  <!-- Создаем контейнер-"вместилище" вкладок и все
  контейнеры-вкладки -->
  <DIV CLASS="tabhost">
    <DIV ID="tab1"><P>Это содержимое первой вкладки.</P></DIV>
    <DIV ID="tab2"><P>Это содержимое второй вкладки.</P></DIV>
    <DIV ID="tab3">
      <P>Это содержимое третьей вкладки.</P>
      <P>И еще рисунок...</P>
      <IMG SRC="image.gif">
    </DIV>
  </DIV>
```

```
    <SCRIPT TYPE="text/javascript">
      // Создаем массивы, содержащие все гиперссылки-корешки и все
      // контейнеры-вкладки соответственно. Эти массивы понадобятся
      // объявленной ранее функции showTab
      var tabs = new Array();
      tabs[0] = document.all["tab1"];
      tabs[1] = document.all["tab2"];
      tabs[2] = document.all["tab3"];
      var tabHs = new Array();
      tabHs[0] = document.all["tabH1"];
      tabHs[1] = document.all["tabH2"];
      tabHs[2] = document.all["tabH3"];
      // Делаем изначально видимой первую вкладку
      showTab(0);
    </SCRIPT>
  </BODY>
</HTML>
```

Как создать слайд-шоу?

Проблема

Я хочу создать на странице слайд-шоу, последовательно выводящее набор изображений без перезагрузки страницы. Как это сделать?

Решение

Это тоже не очень сложно.

1. Создаем на странице "пустое" графическое изображение, т. е. тег `<IMG>` с атрибутом `SRC`, не имеющем значения.
2. Создаем набор гиперссылок для переключения между различными изображениями. Как правило, этих гиперссылок четыре, и осуществляют они переход, соответственно, на первое, на предыдущее, на последующее и на последнее изображение в списке слайд-шоу.
3. Создаем список всех изображений, входящих в слайд-шоу. Этот список проще всего реализовать в виде массива, содержащего интернет-адреса соответствующих файлов (разумеется, в строковом виде).
4. Создаем функции-обработчики события `onClick` созданных на шаге 2 гиперссылок. Эти функции будут выбирать из списка интернет-адрес файла с первым, предыдущим, последующим и последним изображением соответственно и присваивать его свойству `src` созданного на шаге 1 "пустого" изображения `<IMG>`. Также эти функции будут выполнять скрытие неакту-

альных в данный момент гиперссылок (например, если отображается первое изображение в списке, следует скрыть гиперссылки, выполняющие переход на первое и предыдущее изображения).

5. Привязываем созданные на шаге 4 функции-обработчики к событиям `onClick` созданных на шаге 2 гиперссылок.
6. Пишем сценарий, который сразу после загрузки страницы будет выводить первое изображение из списка слайд-шоу.

Опять же, вместо гиперссылок для перемещения между изображениями можно использовать кнопки. Пожалуй, кнопки здесь более уместны.

> ***Народ советует***
>
> Если изображений, входящих в слайд-шоу, немного и они невелики по размеру, можно выполнить их предзагрузку. (О предзагрузке *см. главу 7*.) Это позволит посетителю переключаться между изображениями практически мгновенно, не ожидая окончания их загрузки.

Пример

Далее приведен HTML-код Web-страницы, реализующей слайд-шоу из шести изображений.

```
<HTML>
  <HEAD>
    <TITLE>Слайд-шоу</TITLE>
    <SCRIPT TYPE="text/javascript">
      // Объявляем переменную, хранящую номер изображения, выведенного
      // в данный момент на экран
      var imageIndex = 0;

      // Объявляем функции, выполняющие переключение между изображениями
      function goToFirst()
        {
          imageIndex = 0;
          showImageAndLinks();
        }

      function goToPrevious()
        {
          if (imageIndex > 0) imageIndex--;
          showImageAndLinks();
        }

      function goToNext()
        {
          if (imageIndex < images.length) imageIndex++;
```

```
        showImageAndLinks();
      }

    function goToLast()
      {
        imageIndex = images.length - 1;
        showImageAndLinks();
      }

    // Объявляем функцию, выводящую изображение на экран и скрывающую
    // ненужные в данный момент гиперссылки
    function showImageAndLinks()
      {
        imageHolderObject.src = images[imageIndex];
        firstObject.style.visibility = (imageIndex == 0) ? "hidden" :
        ↳"visible";
        previousObject.style.visibility = firstObject.style.visibility;
        nextObject.style.visibility = (imageIndex == images.length - 1)
        ↳? "hidden" : "visible";
        lastObject.style.visibility = nextObject.style.visibility;
      }
  </SCRIPT>
</HEAD>
<BODY>
  <!-- "Пустое" изображение -->
  <IMG ID="imageHolder" SRC="">
  <!-- Набор гиперссылок для переключения между изображениями -->
  <P><A ID="first" HREF="#" ONCLICK="goToFirst();">&lt;&lt;</A>
   <A ID="previous" HREF="#" ONCLICK="goToPrevious();">&lt;</A>
   <A ID="next" HREF="#" ONCLICK="goToNext();">&gt;</A>
   <A ID="last" HREF="#" ONCLICK="goToLast();">&gt;&gt;</A></P>
  <SCRIPT TYPE="text/javascript">
    // Создаем массив с интернет-адресами всех изображений, входящих
    // в слайд-шоу
    var images = new Array();
    images[0] = "pic1.jpg";
    images[1] = "pic2.jpg";
    images[2] = "pic3.jpg";
    images[3] = "pic4.jpg";
    images[4] = "pic5.jpg";
    images[5] = "lastpic.jpg";
    // Объявляем набор экземпляров объектов, соответствующих "пустому"
    // изображению и гиперссылкам. Они пригодятся объявленным ранее
    // функциям
```

```
      var imageHolderObject = document.all["imageHolder"];
      var firstObject = document.all["first"];
      var previousObject = document.all["previous"];
      var nextObject = document.all["next"];
      var lastObject = document.all["last"];
      // Сразу после загрузки страницы выводим первое изображение
      // в списке слайд-шоу
      goToFirst();
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь для вывода очередного изображения и скрытия ненужных гиперссылок мы использовали отдельную функцию `showImageAndLinks`. Это позволило нам сделать код компактнее.

Народ замечает

Вообще, подобное слайд-шоу, выводящее изображения без перезагрузки страницы, — замечательная вещь. Во-первых, мы можем, как угодно, менять список изображений, входящих в это слайд-шоу, — для этого достаточно изменить код JavaScript, создающий этот список (в приведенном случае — массив `images`). Во-вторых, при переключении на очередное изображение с Web-сервера будет загружен только файл, содержащий это самое изображение; ни Web-страница, ни все сопутствующие ей файлы в этом случае перезагружаться не будут.

Использование внешних баз данных

А сейчас мы рассмотрим использование внешних баз данных. Данные, хранящиеся в таких базах, можно применять для пополнения содержимого Web-страниц, каких-то расчетов, хранения интернет-адресов страниц или изображений для слайд-шоу и пр.

Народ предупреждает!

Во-первых, внешние базы данных поддерживаются только Internet Explorer 4.0 и более новыми его версиями. Во-вторых, такие базы данных в Internet Explorer доступны только для чтения.

Как использовать внешние базы данных?

Проблема

Я слышал, что Internet Explorer поддерживает использование внешних баз данных. Так ли это?

Решение

Да, поддерживает. Конечно, до полноценной системы управления базами данных (СУБД) ему далеко, но простейшие текстовые базы ему по зубам.

Что же такое *текстовая база данных*? Это обычный текстовый файл, содержащий данные, организованные в виде таблицы. Вот так:

```
Java,компилируемый
JavaScript,интерпретируемый
C++,компилируемый
Visual Basic,компилируемый
Perl,интерпретируемый
```

Каждая строка такой "таблицы" называется *записью* и несет сведения о каком-либо одном языке программирования. Столбец же этой таблицы называется *полем* и содержит какую-либо характеристику языка, например, название и категорию (компилируемый или интерпретируемый). Значения разных полей в записи отделяются друг от друга определенным *символом-разделителем*; по умолчанию, как мы видим, это запятая, но его можно и сменить.

Практически всегда в текстовой базе данных записываются имена ее полей. Они помещаются в качестве самой первой записи этой базы (выделены полужирным шрифтом):

```
name,cat
Java,компилируемый
JavaScript,интерпретируемый
C++,компилируемый
Visual Basic,компилируемый
Perl,интерпретируемый
```

Это делается для упрощения доступа к значениям нужных полей.

Помимо имен полей, в первой записи базы можно указать и тип хранящихся в них данных, поместив соответствующее ключевое слово за именем поля и разделив их двоеточием. Например:

```
name:text,cat:text
```

Здесь оба поля базы объявлены текстовыми (ключевое слово `text`).

Полный список типов полей и соответствующих им ключевых слов таков:

- текстовый (`text`) — это тип по умолчанию;
- дата (`date`);
- логический (`boolean`); значению `true` соответствует "yes" или 1, значению `false` — "no" или 0;

- целое число (`int`);
- число с плавающей точкой (`float`).

Итак, мы создали текстовую базу данных и сохранили ее в файле. Как теперь использовать ее на Web-странице?

Прежде всего, нам понадобится особый элемент ActiveX. Он носит название *TDC* (Tabular Data Control, элемент управления табличными данными) и стандартно поставляется в составе Internet Explorer, так что специально его разыскивать и устанавливать на компьютер нам не понадобится.

Элемент TDC создается с помощью вот такого HTML-кода:

```
<OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
↳ID="<Имя>" WIDTH="0" HEIGHT="0">
  <PARAM NAME="DataURL" VALUE="<Имя файла базы данных>">
  <PARAM NAME="UseHeader"
  ↳VALUE="<Считать первую запись списком имен полей>">
</OBJECT>
```

Тег `<OBJECT>`, с помощью которого создается элемент ActiveX, нам знаком по *главе 11*, так что не будем подробно его рассматривать. Отметим только несколько моментов. Во-первых, запомним используемый в нем GUID. Во-вторых, нам обязательно нужно дать ему имя (атрибут `ID`), иначе мы не сможем привязать к нему элементы страницы, которые будут отображать данные из таблицы. В-третьих, крайне желательно задать для него нулевые ширину (атрибут `WIDTH`) и высоту (атрибут `HEIGHT`), чтобы он не присутствовал на Web-странице. (Задать ширину и высоту элемента ActiveX можно и с помощью стилей CSS.)

Обязательными параметрами элемента TDC являются `DateURL`, задающий интернет-адрес текстового файла с базой данных, и `UseHeader`, указывающий, считать ли первую запись базы списком имен ее полей. Остальные параметры являются необязательными. Полный же список параметров элемента TDC приведен в табл. 15.1.

Таблица 15.1. *Параметры элемента TDC*

Параметр	Описание
`CaseSensitive`	Если "1" (значение по умолчанию), при фильтрации и сортировке записей будет учитываться регистр символов. Если "0", регистр учитываться не будет
`CharSet`	Задает кодировку текста для данных текстовой базы. По умолчанию `"windows-1525"` (символы, используемые в большинстве западноевропейских языков). Чтобы нормально отображались символы кириллицы, следует использовать кодировку `"windows-1251"` (кириллица). Также поддерживаются значения `"utf-7"` и `"utf-8"`, соответствующие кодировкам UTF-7 и UTF-8

Таблица 15.1 *(окончание)*

Параметр	Описание
DataURL	Интернет-адрес файла с текстовой базой данных
EscapeChar	Символ, которым в значениях полей таблицы следует предварить символы, являющиеся *служебными* (используемыми для особых целей, например, разделения значений полей). Чаще всего задают значение "обратный слэш" (\)
FieldDelim	Символ-разделитель, используемый для отделения значений полей друг от друга. По умолчанию — запятая
Filter	Критерий фильтрации записей. Подробное описание *см. ниже*
RowDelim	Символ-разделитель, используемый для отделения записей друг от друга. По умолчанию — значение "newline", обозначающее символ перевода строки
Sort	Критерий сортировки записей. Подробное описание *см. далее*
TextQualifier	Символ, в который заключается значение поля, если оно содержит пробелы, запятые и подобные им символы, являющиеся служебными. По умолчанию — двойная кавычка
UseHeader	Если "1", то первая запись базы данных считается списком имен и типов данных полей. Если "0", первая запись базы считается обычными данными, а поля получают имена по умолчанию: "Column1", "Column2" и т. д. Значения по умолчанию у этого параметра нет

Хорошо, элемент TDC мы создали. Как теперь вывести полученные с его помощью из базы данные на страницу?

Для вывода данных из базы мы можем использовать элементы страницы двух типов. Во-первых, это могут быть обычные абзацы, контейнеры <DIV>, поля ввода и пр. — в общем, те элементы, что способны выводить только значение одного поля одной записи базы данных. Во-вторых, это могут быть таблицы — они могут отображать одновременно содержимое нескольких записей базы.

Чтобы любой из упомянутых выше элементов страницы мог отображать данные из базы, нам будет нужно выполнить его *привязку к данным*. Для этого служат несколько особых атрибутов и свойств, перечисленных в табл. 15.2 и поддерживаемых большинством тегов.

Таблица 15.2. *Свойства и атрибуты привязки к данным*

Атрибут	Свойство	Описание
DATAFLD	dataFld	Имя поля базы данных, значение которого отображает элемент страницы, в строковом виде

Таблица 15.2 *(окончание)*

Атрибут	Свойство	Описание
DATAFORMATAS	dataFormatAs	Формат представления данных. Если задано значение "text", данные будут отображаться как обычный текст, а если "HTML" — то с учетом HTML-форматирования (если оно присутствует). Доступно также значение "localized-text", задающее отображение данных как обычный текст, но с учетом национальных установок системы
DATAPAGESIZE	dataPageSize	Задает, сколько записей базы данных будет одновременно отображаться в таблице. Если не указан, отображаются все записи. Поддерживается только таблицами (тегом <TABLE>)
DATASRC	dataSrc	Задает элемент TDC, из которого будут взяты данные

Итак, чтобы привязать к данным, скажем, абзац (тег <P>), нужно использовать атрибуты DATASRC и DATAFLD этого тега. Первый атрибут задает элемент TDC, а второй — поле базы данных. Например:

```
<P DATASRC="#langs" DATAFLD="name"></P>
```

Заметим, что, во-первых, в этом случае имя элемента TDC должно предваряться значком "решетка" (#), а сам привязываемый к данным тег <P> не должен иметь никакого содержимого.

Если же мы хотим вывести содержимое базы данных в таблице, то должны будем сделать следующее. Сначала мы создадим тег <TABLE> и в нем с помощью атрибута DATASRC укажем элемент TDC. Также мы можем указать в этом теге атрибут DATAPAGESIZE, задающий количество одновременно отображаемых записей. После этого мы обязательно создадим секцию тела (тег <TBODY>), а в ней — строку (тег <TR>) с набором ячеек (теги <TD>). В каждую из этих ячеек мы вложим тег <P> или <DIV>, в котором с помощью атрибута DATAFLD зададим имя поля, значения которых будут отображаться в данной ячейке. Например:

```
<TABLE DATASRC="#langs" DATAPAGESIZE="10">
  <TBODY>
    <TR>
      <TD><DIV DATAFLD="name"></DIV></TD>
      <TD><DIV DATAFLD="cat"></DIV></TD>
    </TR>
  </TBODY>
</TABLE>
```

При этом секция тела таблицы должна включать только одну строку, содержащую определение ячеек, в которых будут выводиться данные из базы. Web-обозреватель сам "размножит" эту строку, чтобы отобразить либо все записи базы данных, либо столько, сколько указано в атрибуте DATAPAGESIZE.

> ***Народ предупреждает!***
>
> Элемент TDC, извлекающий данные из базы, должен находиться в HTML-коде страницы перед любым ее элементом, выводящим эти данные.
>
> Использовать для вывода данных из базы непосредственно теги <TD> невозможно.

Мы также можем создать в таблице секции "шапки" и "поддона":

```
<TABLE DATASRC="#langs" DATAPAGESIZE="10">
  <THEAD>
    <TR>
      <TH>Язык</TH>
      <TH>Категория</TH>
    </TR>
  </THEAD>
  </TBODY>
    <TR>
      <TD><DIV DATAFLD="name"></DIV></TD>
      <TD><DIV DATAFLD="cat"></DIV></TD>
    </TR>
  </TBODY>
</TABLE>
```

При этом для вывода данных из базы будет все равно использоваться только секция тела. Секции "шапки" и "поддона" для этого использованы не будут.

Мы можем привязать таблицу к данным и немного по-другому:

```
<TABLE DATAPAGESIZE="10">
  </TBODY>
    <TR>
      <TD><DIV DATASRC="#langs" DATAFLD="name"></DIV></TD>
      <TD><DIV DATASRC="#langs" DATAFLD="cat"></DIV></TD>
    </TR>
  </TBODY>
</TABLE>
```

т. е. указав атрибут DATASRC прямо в тегах <DIV>, помещенных в ячейки, в которых будут отображаться данные из базы. Но такой вариант привязки таблиц к данным несколько громоздок.

Пример

Давайте создадим такую текстовую базу данных:

```
name:text,cat:text
Java,компилируемый
JavaScript,интерпретируемый
C++,компилируемый
Visual Basic,компилируемый
Perl,интерпретируемый
Borland Delphi,компилируемый
```

Сохраним ее в файле jspsLangs.txt.

> ***Народ предупреждает!***
>
> Если создать в базе данных пустую строку, она будет истолкована элементом TDC как *пустая запись* (запись, поля которой не содержат значений). Это относится и к пустой строке в конце текстового файла, являющегося базой данных.

После этого напишем Web-страницу для вывода данных из этой базы. Ее HTML-код будет таков:

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ↳ID="langs" WIDTH="0" HEIGHT="0">
      <PARAM NAME="DataURL"   VALUE="jspsLangs.txt">
      <PARAM NAME="UseHeader" VALUE="1">
    </OBJECT>
    <TABLE DATASRC="#langs">
      <THEAD>
        <TR>
          <TH>Язык</TH>
          <TH>Категория</TH>
        </TR>
      </THEAD>
      <TBODY>
        <TR>
          <TD><DIV DATAFLD="name"></DIV></TD>
          <TD><DIV DATAFLD="cat"></DIV></TD>
        </TR>
      </TBODY>
    </TABLE>
```

```
  </BODY>
</HTML>
```

Если теперь мы сохраним эту страницу в файле и откроем ее в Web-обозревателе, таблица заполнится списком языков программирования, взятым из базы данных jspsLangs.txt.

Как программно управлять элементом TDC?

Проблема

Поддерживает ли элемент TDC какие-либо методы и свойства для перемещения по базе данных и выборке данных из нее?

Решение

Конечно, поддерживает. Давайте их рассмотрим.

Прежде всего, нужно сказать, что сам элемент TDC поддерживает набор свойств, аналогичных параметрам, перечисленным в табл. 15.1. Эти свойства имеют те же имена, что и соответствующие им параметры. Единственное — вместо строковых значений "1" и "0" для задания логических величин следует применять привычные нам `true` и `false`.

Далее, элемент TDC поддерживает свойство `recordset`, предоставляющее доступ к экземпляру объекта, который соответствует так называемому *набору записей*. Набор записей — это программное представление самой базы данных, со всеми ее полями и записями. Пользуясь набором записей, мы можем перемещаться от записи к записи и получать значения ее полей.

Для перемещения от записи к записи мы можем использовать методы объекта `recordset`. Методы `MovePrevious` и `MoveNext` выполняют перемещение соответственно на предыдущую и последующую запись базы данных. А выполнить перемещение на первую и последнюю запись базы можно вызовом методов `MoveFirst` и `MoveLast` соответственно. Все эти методы не принимают параметров и не возвращают значений.

Особые свойства `bof` и `eof` объекта `recordset` также будут нам полезны. Свойство `bof` возвращает `true`, если мы находимся на первой записи базы. Если же мы переместились на последнюю запись, то свойство `eof` вернет значение `true`. Таким образом, мы всегда будем знать, на какой записи базы данных мы находимся.

Народ советует

Если мы находимся на первой записи базы данных и вызываем метод `MovePrevious`, то получим сообщение об ошибке. То же самое будет, если переместиться на последнюю запись базы и вызвать метод `MoveNext`. Так что при написании сценариев, работающих с элементом TDC, перед собственно пере-

мещением на другую запись всегда следует выполнять проверку значений свойств `bof` и `eof`.

Еще объект `recordset` поддерживает коллекцию `Fields`, содержащую все поля базы данных. Мы можем получить доступ к любому полю базы как по ее номеру, так и по имени. А для получения значения поля мы должны будем обратиться к свойству `Value` соответствующего поля.

Например, следующее выражение поместит в переменную `cat` значение поля `cat` базы данных из предыдущего примера.

```
var cat = langsObject.recordset.Fields("cat").Value;
```

Здесь `langsObject` — элемент TDC, с помощью которого мы получаем доступ к нашей базе данных. Также обратим внимание на то, что имя поля указывается в этом случае не в квадратных, а в круглых скобках — это обязательно.

Народ предупреждает!

Мы можем получить значение любого поля только той записи, на которой находимся в данный момент *(текущей записи)*. Если нам понадобится значение какого-либо поля другой записи, нам придется переместиться на нее, используя уже знакомые нам методы `MoveFirst`, `MovePrevious`, `MoveNext` и `MoveLast`.

Кроме того, нам пригодятся еще два свойства объекта `recordset`. Свойство `AbsolutePosition` возвращает номер записи, являющейся в данный момент текущей. А свойство `RecordCount` возвращает общее количество записей в базе данных.

Кроме того, элемент TDC поддерживает несколько специфических событий, перечисленных в табл. 15.3.

Таблица 15.3. *Специфические события элемента TDC*

Событие	Описание
`onDataAvailable`	Наступает всякий раз, когда набор записей готов предоставить новые данные
`onDatasetChanged`	Наступает, когда параметры набора записей изменяются, например, после задания критериев фильтрации или сортировки
`onDatasetComplete`	Наступает после того, как набор записей передаст все доступные данные
`onRowEnter`	Наступает после перехода на текущую запись базы данных
`onRowExit`	Наступает перед переходом с текущей записи на другую

Эти события можно использовать по-разному. Например, событие `onRowEnter` можно использовать для вывода информации, которая не хранится в базе данных, а рассчитывается на основе ее содержимого.

Пример 1

Далее приведен HTML-код Web-страницы, выводящей содержимое текущей записи базы данных jspsLangs.txt из предыдущего примера. Для перехода между записями используется набор гиперссылок.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ↳ID="langs" WIDTH="0" HEIGHT="0">
      <PARAM NAME="DataURL" VALUE="jspsLangs.txt">
      <PARAM NAME="UseHeader" VALUE="1">
    </OBJECT>
    <P>Язык программирования: <SPAN DATASRC="#langs"
    ↳DATAFLD="name"></SPAN></P>
    <P>Категория: <SPAN DATASRC="#langs" DATAFLD="cat"></SPAN></P>
    <P><A HREF="#" ONCLICK="if (!recObject.bof) recObject.MoveFirst();">
    ↳&lt;&lt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.bof) recObject.MovePrevious();">
    ↳&lt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.eof) recObject.MoveNext();">
    ↳&gt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.eof) recObject.MoveLast();">
    ↳&gt;&gt;</A></P>
    <SCRIPT TYPE="text/javascript">
      var langsObject = document.all["langs"];
      var recObject = langsObject.recordset;
    </SCRIPT>
  </BODY>
</HTML>
```

Заметим, что перед вызовом любого метода объекта `recordset`, выполняющего перемещение между записями, мы проверяем значение свойства `bof` или `eof`. Это позволит нам избежать сообщений об ошибках.

Пример 2

Немного исправим HTML-код из предыдущего примера таким образом, чтобы в случае выбора компилируемого языка программирования на странице появлялось слово "Компилируемый".

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ↳ID="langs" WIDTH="0" HEIGHT="0"
    ↳ONROWENTER="compObject.style.visibility =
    ↳(langsObject.recordset.Fields('cat').Value == 'компилируемый') ?
    ↳'visible' : 'hidden'">
      <PARAM NAME="DataURL" VALUE="jspsLangs.txt">
      <PARAM NAME="UseHeader" VALUE="1">
    </OBJECT>
    <P>Язык программирования: <SPAN DATASRC="#langs"
    ↳DATAFLD="name"></SPAN></P>
    <P ID="comp">Компилируемый</P>
    <P><A HREF="#" ONCLICK="if (!recObject.bof) recObject.MoveFirst();">
    ↳&lt;&lt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.bof) recObject.MovePrevious();">
    ↳&lt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.eof) recObject.MoveNext();">
    ↳&gt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.eof) recObject.MoveLast();">
    ↳&gt;&gt;</A></P>
    <SCRIPT TYPE="text/javascript">
      var langsObject = document.all["langs"];
      var recObject = langsObject.recordset;
      var compObject = document.all["comp"];
    </SCRIPT>
  </BODY>
</HTML>
```

Здесь для скрытия и отображения надписи "Компилируемый" мы используем обработчик события `onRowEnter` элемента TDC.

Как отфильтровать нужные мне записи и отсортировать их?

Проблема

Все уважающие себя СУБД позволяют отфильтровывать нужные пользователю записи и сортировать их по заданным критериям. Неужели разработчики из фирмы Microsoft этого в Internet Explorer не предусмотрели?

Решение

Предусмотрели, да еще как! Давайте сами посмотрим.

Задать критерий *фильтрации* (т. е. отбора нужных записей) мы можем с помощью свойства `Filter` элемента TDC. (Существует также одноименный параметр, задаваемый тегом `<PARAM>`.) Критерий фильтрации задается в виде выражения сравнения в строковом виде, например:

```
langsObject.Filter = "cat = 'компилируемые'";
```

В табл. 15.4 перечислены все операторы сравнения и логические операторы, которые мы можем использовать в таких выражениях.

Таблица 15.4. *Операторы сравнения и логические операторы, используемые в выражениях сравнения при задании критериев фильтрации записей*

Оператор	Описание
<	Меньше
>	Больше
=	Равно
<=	Меньше или равно
>=	Больше или равно
<>	Не равно
&	Логическое И
|	Логическое ИЛИ

Народ предупреждает!

Заметим, что для создания оператора равенства в критериях фильтрации используется один знак "равно", а не два, как в JavaScript.

Чтобы вообще отключить фильтрацию, достаточно присвоить свойству `Filter` элемента TDC пустую строку.

Для задания критерия сортировки предназначено свойство `Sort` (и одноименный параметр) элемента TDC. Этому свойству присваивается строка, содержащая список имен полей, по которым должна выполняться сортировка, разделенных точкой с запятой. Например:

```
langsObject.Sort = "cat;name";
```

После выполнения этого выражения база данных будет отсортирована по полю `cat`. Записи же, содержащие одинаковые значения поля `cat`, будут отсортированы также и по полю `name`.

По умолчанию сортировка выполняется по возрастанию значения заданного поля. Чтобы задать сортировку по убыванию, нужно перед именем поля поставить знак "минус":

```
langsObject.Sort = "cat;-name";
```

В этом случае сортировка по полю `name` будет выполняться по убыванию значения этого поля.

Опять же, чтобы вообще отключить сортировку, достаточно присвоить свойству `Sort` элемента TDC пустую строку.

После задания критериев фильтрации и (или) сортировки нужно вызвать метод `Reset` элемента TDC, который выполнит обновление всех элементов страницы, привязанных к данным. Этот метод не принимает параметров и не возвращает значения.

Пример

Далее приведен HTML-код Web-страницы, выводящей список языков программирования из базы данных jspsLangs.txt. Посетитель может выбирать категорию выводимых в списке языков с помощью переключателей **Компилируемые** и **Интерпретируемые**.

```
<HTML>
  <HEAD>
    <TITLE>Языки программирования</TITLE>
    <SCRIPT TYPE="text/javascript">
      function setFilter(pCat)
        {
          langsObject.Filter = "cat = \"" + pCat + "\"";
          langsObject.Reset();
        }
    </SCRIPT>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ↳ID="langs" WIDTH="0" HEIGHT="0">
      <PARAM NAME="DataURL" VALUE="jspsLangs.txt">
      <PARAM NAME="UseHeader" VALUE="1">
      <PARAM NAME="Sort" VALUE="name">
    </OBJECT>
    <FORM>
      <P><INPUT TYPE="radio" ID="sort" NAME="sort"
      ↳ONCLICK="setFilter('компилируемый');" CHECKED> Компилируемые</P>
```

```
      <P><INPUT TYPE="radio" ID="sort" NAME="sort"
      ↳ONCLICK="setFilter('интерпретируемый');"> Интерпретируемые</P>
    </FORM>
    <TABLE DATASRC="#langs">
      <THEAD>
        <TR>
          <TH>Язык</TH>
          <TH>Категория</TH>
        </TR>
      </THEAD>
      <TBODY>
        <TR>
          <TD><DIV DATAFLD="name"></DIV></TD>
          <TD><DIV DATAFLD="cat"></DIV></TD>
        </TR>
      </TBODY>
    </TABLE>
    <SCRIPT TYPE="text/javascript">
      var langsObject = document.all["langs"];
      setFilter("компилируемый");
    </SCRIPT>
  </BODY>
</HTML>
```

Заметим, что критерий сортировки задается прямо в коде HTML (параметр `Sort`), создающем элемент TDC, т. к. он никогда не меняется (в отличие от критерия фильтрации).

Можно ли хранить в базе данных фрагменты страницы?

Проблема

Да, хранить данные в базе — неплохое решение. Но можно ли поместить туда целые фрагменты HTML-кода, а потом извлекать их и помещать на страницу?

Решение

Конечно, можно! Специально для этого в Internet Explorer предусматривается параметр `DATAFORMATAS`, поддерживаемый всеми тегами, которые можно привязать к данным. Значение `"HTML"` этого параметра предписывает Internet Explorer обрабатывать извлекаемые из поля данные как код HTML. (Значение по умолчанию — `"text"` — указывает ему считать эти данные обычным текстом.)

Пример

Давайте создадим "развернутое" слайд-шоу, представляющее, помимо самих изображений, еще и примечания к ним. Сначала создадим вот такую базу данных:

```
content
<IMG SRC="1.jpg"><P>Начальный кадр фильма &quot;33 несчастья&quot;.</P>
<IMG SRC="2.jpg"><P>Приезд мистера По.</P>
<IMG SRC="3.jpg"><P>Печальное известие для Бодлеров...</P>
<IMG SRC="4.jpg"><P>Дом профессора Монтгомери.</P>
<IMG SRC="5.jpg"><P>Дом над водой.</P>
<IMG SRC="6.jpg"><P>Последний кадр. Что-то ждет их впереди?..</P>
```

Автор книги использовал для слайд-шоу собственноручно "снятые" кадры фильма "33 несчастья". (Разумеется, можно использовать любые другие изображения и любые другие текстовые примечания к ним.) Первой записью базы данных указано имя единственного ее поля — `content`.

Сохраним готовую базу данных в файле jspsSlideShow.txt и приступим к созданию страницы, отображающей это слайд-шоу. Вот ее HTML-код:

```
<HTML>
  <HEAD>
    <TITLE>Слайд-шоу</TITLE>
  </HEAD>
  <BODY>
    <OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ↳ID="slideshow" WIDTH="0" HEIGHT="0">
      <PARAM NAME="CharSet" VALUE="windows-1251">
      <PARAM NAME="DataURL" VALUE="jspsSlideShow.txt">
      <PARAM NAME="UseHeader" VALUE="1">
    </OBJECT>
    <DIV DATASRC="#slideshow" DATAFLD="content"
    ↳DATAFORMATAS="HTML"></DIV>
    <P><A HREF="#" ONCLICK="if (!recObject.bof) recObject.MoveFirst();">
    ↳&lt;&lt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.bof) recObject.MovePrevious();">
    ↳&lt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.eof) recObject.MoveNext();">
    ↳&gt;</A> 
    <A HREF="#" ONCLICK="if (!recObject.eof) recObject.MoveLast();">
    ↳&gt;&gt;</A></P>
    <SCRIPT TYPE="text/javascript">
      var slideShowObject = document.all["slideshow"];
```

```
      var recObject = slideShowObject.recordset;
    </SCRIPT>
  </BODY>
</HTML>
```

Единственно, что здесь нужно отметить, — это задание кодировки текста с помощью параметра `CharSet`. Здесь мы задали кодировку `"windows-1251"` (стандартная "кириллическая" кодировка Windows). Если этого не сделать, символы кириллицы, встречающиеся в базе данных, могут отображаться некорректно.

Народ советует

Вообще, кодировку текста в элементе TDC следует указывать всегда. Как говорится, во избежание проблем.

Народ советует

Желающие узнать побольше об элементе TDC и привязке элементов страниц к данным могут обратиться к соответствующему разделу MSDN по адресу **http://msdn.microsoft.com/library/default.asp?url=/workshop/author/databind/data_binding_node_entry.asp**.

Как создавать HTML-приложения?

Проблема

Говорят, что Internet Explorer поддерживает какие-то HTML-приложения. Что это такое, чем они хороши и как их создавать?

Решение

HTML-приложения — это клиентские Windows-приложения, написанные с использованием HTML, CSS и JavaScript в виде Web-страницы и выполняющиеся Internet Explorer. Они отображаются в своем собственном окне, параметры которого мы можем менять в довольно широких пределах, выводятся в списке Диспетчера задач Windows и могут показываться на панели задач. Также они имеют свое имя, под которым и отображаются в списке задач, и, возможно, свой значок, показываемый в Проводнике.

HTML-приложения также отличаются от обычных Web-страниц тем, что Internet Explorer считает их содержимое изначально безопасным. Это значит, что мы можем помещать в HTML-приложения элементы ActiveX, и они будут выводиться без всяких предупреждений.

Народ предупреждает!

HTML-приложения поддерживаются только Internet Explorer 5.0 и более новыми его версиями.

От обычных Web-страниц HTML-приложения отличаются, во-первых, тем, что их файл имеет расширение hta, а не htm[l], а во-вторых, присутствием в их секции заголовка (теге <HEAD>) особого парного тега HTA:APPLICATION, атрибуты которого задают различные параметры данного приложения. Давайте-ка его рассмотрим.

Итак, парный тег HTA:APPLICATION. Именно он превращает обычную Web-страницу в HTML-приложение. Располагается он в секции заголовка Web-страницы (т. е. внутри парного тега <HEAD>) и не имеет в себе никакого содержимого. Зато он имеет множество атрибутов, с помощью которых задаются различные параметры HTML-приложения. Эти атрибуты перечислены в табл. 15.5.

Таблица 15.5. *Атрибуты тега* `HTA:APPLICATION`

Атрибут	Описание
APPLICATIONNAME	Задает имя HTML-приложения в строковом виде. Под этим именем приложение будет присутствовать в списке Диспетчера задач и панели задач Windows
BORDER	Задает вид рамки окна HTML-приложения. Доступны значения: "thick" (толстая рамка с возможностью изменения размера окна; значение по умолчанию), "dialog" (толстая рамка без возможности изменения размеров окна), "thin" (тонкая рамка без возможности изменения размеров окна) и "none" (рамка отсутствует)
BORDERSTYLE	Задает вид обрамления содержимого окна HTML-приложения, т. е. самой страницы. Доступны значения: "normal" (обычное; значение по умолчанию), "raised" (трехмерное приподнятое), "sunken" (трехмерное вдавленное), "complex" (трехмерное приподнятое и вдавленное) и "static" (трехмерное, используемое, в основном, в приложениях, не требующих пользовательского ввода)
CAPTION	Если "yes" (значение по умолчанию), окно HTML-приложения будет содержать заголовок, если "no" — не будет
CONTEXTMENU	Если "yes" (значение по умолчанию), при щелчке правой кнопкой мыши по странице, являющейся HTML-приложением, появится контекстное меню, если "no" — не появится
ICON	Задает имя файла значка, используемого Проводником Windows для показа файла HTML-приложения. Используется обычный файл значков Windows с расширением ico; размеры значка — 32×32 пиксела
INNERBORDER	Если "yes" (значение по умолчанию), между границами окна HTML-приложения и содержимым выводимой в нем Web-страницы будет отображена рамка, если "no" — не будет

Таблица 15.5 *(окончание)*

Атрибут	Описание
MAXIMIZEBUTTON	Если "yes" (значение по умолчанию), заголовок окна HTML-приложения будет содержать кнопку максимизации, если "no" — не будет
MINIMIZEBUTTON	Если "yes" (значение по умолчанию), заголовок окна HTML-приложения будет содержать кнопку минимизации, если "no" — не будет
NAVIGABLE	Если "no" (значение по умолчанию), то при щелчке на гиперссылках, содержащихся на Web-странице, соответствующие им страницы будут загружены в новое окно Web-обозревателя, если "yes" — в то же самое окно
SCROLL	Если "yes", в окне HTML-приложения всегда будут присутствовать полосы прокрутки, если "no" — не будут присутствовать в любом случае, если "auto" — будут присутствовать, только если содержимое Web-страницы не помещается в окне
SCROLLFLAT	Если "no" (значение по умолчанию), полосы прокрутки будут иметь трехмерный вид, если "yes" — плоский вид
SELECTION	Если "yes" (значение по умолчанию), содержимое Web-страницы, являющейся HTML-приложением, можно выделить, если "no" — нельзя
SHOWINTASKBAR	Если "yes" (значение по умолчанию), HTML-приложение будет присутствовать в панели задач Windows, если "no" — не будет. В списке Диспетчера задач оно будет присутствовать в любом случае
SINGLEINSTANCE	Если "no" (значение по умолчанию), пользователь может запустить одновременно несколько копий этого HTML-приложения, если "yes" — только одну копию
SYSMENU	Если "yes" (значение по умолчанию), заголовок окна HTML-приложения будет содержать системное меню, если "no" — не будет
VERSION	Задает версию HTML-приложения в строковом виде
WINDOWSTATE	Задает начальное состояние окна HTML-приложения, которое оно примет сразу же после открытия. Доступны значения: "normal" (обычное; значение по умолчанию), "minimize" (минимизированное) и "maximize" (максимизированное)

Само содержимое HTML-приложения помещается, как и в случае обычной Web-страницы, в секции тела (в теге <BODY>). Ведь, как мы помним, HTML-приложение — фактически обычная Web-страница.

Народ предупреждает!

Если HTML-приложение содержит фреймы, в которых отображаются другие Web-страницы, их содержимое Web-обозреватель считает небезопасным. (Со всеми вытекающими последствиями; так, если страница, открытая во фрейме, содержит элемент ActiveX, Web-обозреватель выдаст пользователю предупреждение о возможно небезопасном содержимом.) Чтобы заставить Web-обозреватель "проникнуться доверием" к открытой во фрейме странице, нужно поместить в создающий этот фрейм тег `<FRAME>` или `<IFRAME>` атрибут `APPLICATION` и присвоить этому атрибуту значение `"yes"`.

Кроме того, тег `HTA:APPLICATION` поддерживает хорошо знакомый нам атрибут `ID`, с помощью которого задается его имя. Так что мы можем обратиться к этому тегу из сценария и получить заданные в нем сведения об HTML-приложении, например, его название и версию. Для этого нужно использовать свойства, перечисленные в табл. 15.6. Все эти свойства доступны только для чтения.

Таблица 15.6. *Свойства тега* `HTA:APPLICATION`

Свойство	Описание
`applicationName`	Возвращает имя HTML-приложения в строковом виде
`border`	Возвращает вид рамки окна HTML-приложения в строковом виде. Все доступные значения перечислены в табл. 15.5
`borderStyle`	Возвращает вид обрамления содержимого окна HTML-приложения, т. е. самой страницы, в строковом виде. Все доступные значения перечислены в табл. 15.5
`caption`	Возвращает `true`, если окно HTML-приложения содержит заголовок, и `false` в противном случае
`commandLine`	Возвращает в строковом виде параметры командной строки, использованные при запуске HTML-приложения
`contextMenu`	Возвращает `"yes"`, если в HTML-приложении разрешен вывод контекстного меню при щелчке правой кнопкой мыши, и `"no"` в противном случае
`icon`	Возвращает имя файла значка, используемого Проводником Windows для вывода файла HTML-приложения
`innerBorder`	Возвращает `"yes"`, если между границами окна HTML-приложения и содержимым отображаемой в нем Web-страницы присутствует рамка, и `"no"` в противном случае
`maximizeButton`	Возвращает `true`, если заголовок окна HTML-приложения содержит кнопку максимизации, и `false` в противном случае
`minimizeButton`	Возвращает `true`, если заголовок окна HTML-приложения содержит кнопку минимизации, и `false` в противном случае

Таблица 15.6 *(окончание)*

Свойство	Описание
navigable	Возвращает "yes", если при щелчке на гиперссылках, содержащихся на Web-странице, соответствующие им страницы загружаются в то же самое окно Web-обозревателя, и "no", если они загружаются в новое окно
scroll	Возвращает "yes", если в окне HTML-приложения всегда присутствуют полосы прокрутки, "no", если они не присутствуют в любом случае, и "auto", если они присутствуют, только когда содержимое Web-страницы не помещается в окне
scrollFlat	Возвращает "yes", если полосы прокрутки имеют плоский вид, и "no", если они имеют трехмерный вид
selection	Возвращает "yes", если содержимое Web-страницы можно выделить, и "no", если нельзя
showInTaskBar	Возвращает true, если HTML-приложение присутствует в панели задач Windows, и false, если не присутствует
singleInstance	Возвращает true, если пользователь может запустить только одну копию этого HTML-приложения, и false, если он может запустить одновременно несколько копий
sysMenu	Возвращает true, если заголовок окна HTML-приложения содержит системное меню, и false, если не содержит
version	Возвращает версию HTML-приложения в строковом виде
windowState	Возвращает в строковом виде начальное состояние окна HTML-приложения, которое оно приняло сразу же после открытия. Все доступные значения перечислены в табл. 15.5

Обратим внимание, что некоторые свойства, перечисленные в табл. 15.6, возвращают логические значения true и false, а другие — строковые "yes" и "no".

Как правило, HTML-приложения распространяются так же, как обычные приложения Windows, — через Интернет в архивах или дистрибутивных файлах. Также возможно их распространение через Интернет и как обычных Web-страниц: HTML-приложение публикуется на Web-сервере и открывается щелчком по гиперссылке на другой Web-странице или набором интернет-адреса в строке адреса Web-обозревателя.

Пример

Далее приведен код небольшого HTML-приложения, выполняющего преобразование размеров из дюймов в миллиметры.

```
<HTML>
  <HEAD>
    <TITLE>Конвертор</TITLE>
    <HTA:APPLICATION ID="app" APPLICATIONNAME="Конвертор" BORDER="dialog"
    ↳ICON="jspsConvertorIcon.ico" MAXIMIZEBUTTON="no" SCROLL="no"
    ↳VERSION="1.0">
    </HTA:APPLICATION>
    <SCRIPT TYPE="text/javascript">
      function convert()
        {
          var inchesObject = document.all["inches"];
          var millimetersObject = document.all["millimeters"];
          if (!isNaN(inchesObject.value))
            millimetersObject.firstChild.nodeValue = inchesObject.value *
            ↳25.4;
        }

      function showAbout()
        {
          var appObject = document.all["app"];
          window.alert(appObject.applicationName + " " +
          ↳appObject.version);
        }

      window.resizeTo(500, 350);
    </SCRIPT>
  </HEAD>
  <BODY>
    <H1>Конвертор</H1>
    <P>Преобразование размеров из дюймов в миллиметры.</P>
    <P>Введите в поле ввода размер в дюймах и нажмите кнопку
    &amp;quot;Преобразовать&amp;quot;.</P>
    <FORM>
      <P>Дюймы: <INPUT TYPE="text" ID="inches" NAME="inches"
      ↳VALUE="0"></P>
      <P><INPUT TYPE="button" VALUE="Преобразовать"
      ↳ONCLICK="convert();"></P>
    </FORM>
    <P ID="millimeters">0</P>
    <FORM>
      <P><INPUT TYPE="button" VALUE="О программе"
      ↳ONCLICK="showAbout();">&amp;nbsp;<INPUT TYPE="button" VALUE="Выход"
      ↳ONCLICK="window.close();"></P>
    </FORM>
  </BODY>
</HTML>
```

Это HTML-приложение использует в качестве значка файл jspsConvertorIcon.ico. Хоть значок для HTML-приложения и не обязателен, его наличие требует хороший стиль программирования.

А теперь — важный момент! Сразу же после открытия окна HTML-приложения мы изменяем его размеры вызовом метода `resizeTo` объекта `window` (об этом методе говорилось в *главе 4*). Без этого не обойтись, поскольку окно Web-обозревателя после открытия имеет случайные размеры, и может быть, что оно окажется слишком маленьким для того, чтобы вместить содержимое приложения. (Это особенно важно, поскольку мы отключили полосы прокрутки, присвоив значение `"no"` атрибуту `SCROLL` тега `HTA:APPLICATION`.) Но даже если окно окажется достаточно большим, в нем может появиться свободное пространство, не занятое ничем, что тоже некрасиво.

Поскольку содержимое нашего приложения имеет, в общем-то, фиксированные размеры, мы задали для окна приложения "диалоговую" рамку — толстую и без возможности изменить размеры окна. Для этого мы присвоили атрибуту `BORDER` тега `HTA:APPLICATION` значение `"dialog"`. Далее, мы отключили кнопку максимизации, присвоив значение `"no"` атрибуту `MAXIMIZEBUTTON` этого же тега. Ну, и отключили полосы прокрутки, о чем уже говорилось.

Далее, для нашего приложения мы задали версию 1.0 (значение атрибута `VERSION` тега `HTA:APPLICATION`). Этого тоже требует хороший стиль программирования — всегда задавать версию приложения. Иначе пользователь не сможет выяснить, последнюю ли версию он использует.

И, наконец, мы предусмотрели в приложении кнопки **О программе** и **Выход**. Их наличия требует все тот же хороший стиль программирования. Пользователь, во-первых, должен знать, что это за приложение и какова его версия, а во-вторых, должен иметь возможность закрыть его разными способами — не только щелчком по кнопке закрытия в заголовке окна.

Народ советует

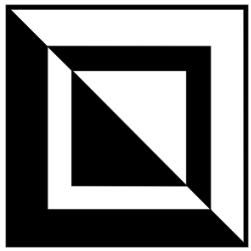
Не буду пересказывать, что написал автор. Лучше сами прочитайте последние несколько абзацев — это важно.

Что дальше?

Что ж, сложные вопросы Web-программирования оказались не такими уж и сложными в изложении народа... Так?

А если так, идем дальше. Последняя глава книги будет посвящена отладке Web-сценариев. Точнее, тем весьма небогатым средствам, которые предоставляют для этого Web-обозреватели.

ГЛАВА 16



Отладка Web-сценариев

Ну, вот и последняя глава! В ней народ попытает Web-обозреватели на предмет средств, с помощью которых можно посмотреть, как выполняются сценарии, и выловить возможные ошибки. Средства эти — увы! — небогаты, но даже они будут нам полезны.

Как найти синтаксические ошибки?

Проблема

Я подозреваю, что в моих сценариях присутствуют *синтаксические ошибки* (ошибки в написании ключевых слов, операторов и выражений языка программирования). Как мне их найти?

Решение (Internet Explorer)

Чтобы включить в Internet Explorer вывод сообщений об ошибках, нужно выбрать пункт **Свойства обозревателя** в меню **Сервис**. На экране появится диалоговое окно **Свойство обозревателя**. Далее следует переключиться на вкладку **Дополнительно** этого окна и прокрутить занимающий его целиком иерархический список, чтобы добраться до "ветви" **Обзор** (рис. 16.1). После этого останется только включить расположенные в этой ветви флажки **Выводить подробные сообщения об ошибках http** и **Показывать уведомление о каждой ошибке сценария** и нажать кнопку **OK**.

> ***Народ замечает***
>
> Вообще-то, эти флажки могут быть включены по умолчанию... Но проверить не помешает. Это, кстати, справедливо и для других Web-обозревателей.

После этого, если в сценарии встретится синтаксическая ошибка, Internet Explorer выведет на экран окно-предупреждение, показанное на рис. 16.2.

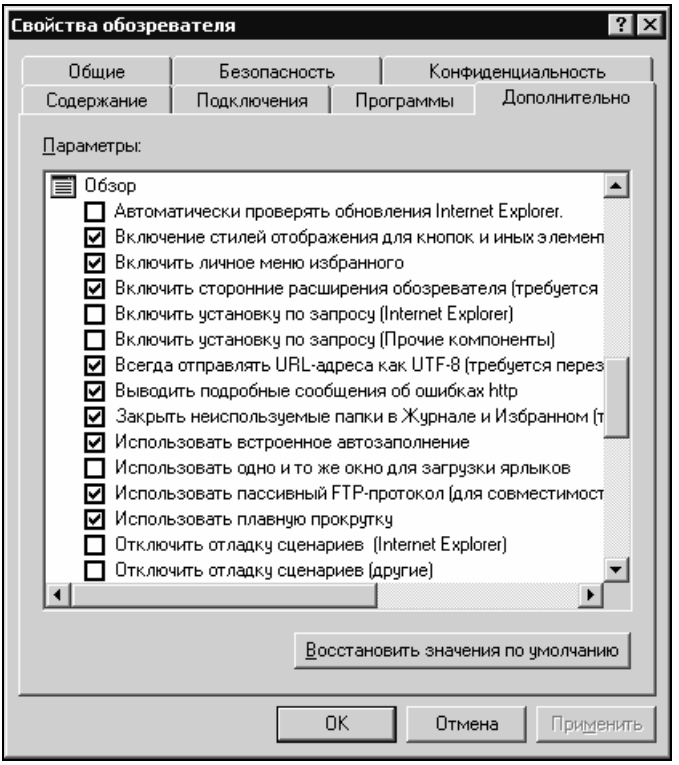


Рис. 16.1. Диалоговое окно **Свойства обозревателя** Internet Explorer (вкладка **Дополнительно**)

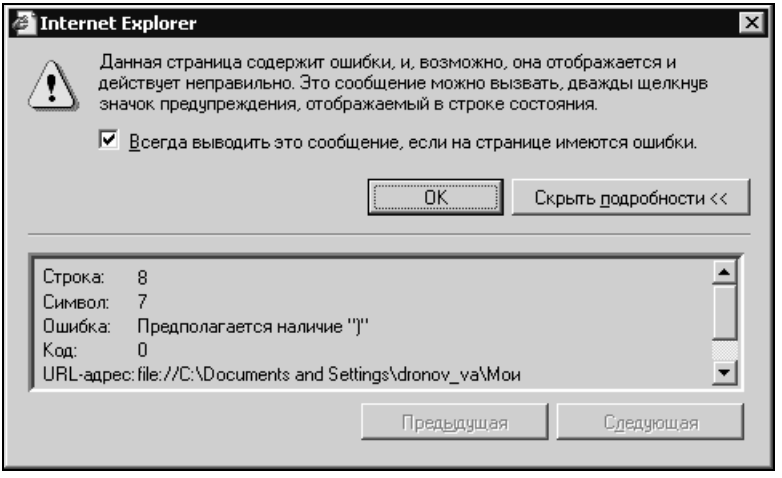


Рис. 16.2. Окно-предупреждение о синтаксической ошибке, выводимое Internet Explorer

Как видим, в текстовом поле в нижней половине окна отображаются весьма подробные сведения об ошибке, в том числе ее краткое описание и номер строки и символа в коде Web-страницы, где она встретилась. В случае, показанном на рис. 16.2, это отсутствие закрывающей скобки.

Народ предупреждает!

Надо сказать, что Internet Explorer, как и другие Web-обозреватели, зачастую показывает номер строки и символа, в которых встретилась ошибка, весьма приблизительно. Поэтому следует просмотреть все "близлежащие" строки кода JavaScript — возможно, ошибка присутствует именно в них, а не в указанной Web-обозревателем строке.

После нажатия кнопки **OK** показанного на рис. 16.2 окна-предупреждения выполнение кода сценария продолжится. Хотя, разумеется, страница после этого вряд ли будет работать правильно...

Решение (Opera)

Чтобы включить вывод предупреждений об ошибках Opera, нужно выбрать пункт **Настройки** в меню **Инструменты** и переключиться на вкладку **Дополнительно** появившегося на экране диалогового окна **Настройки** (рис. 16.3). Далее следует нажать кнопку **Параметры JavaScript** этой вклад-

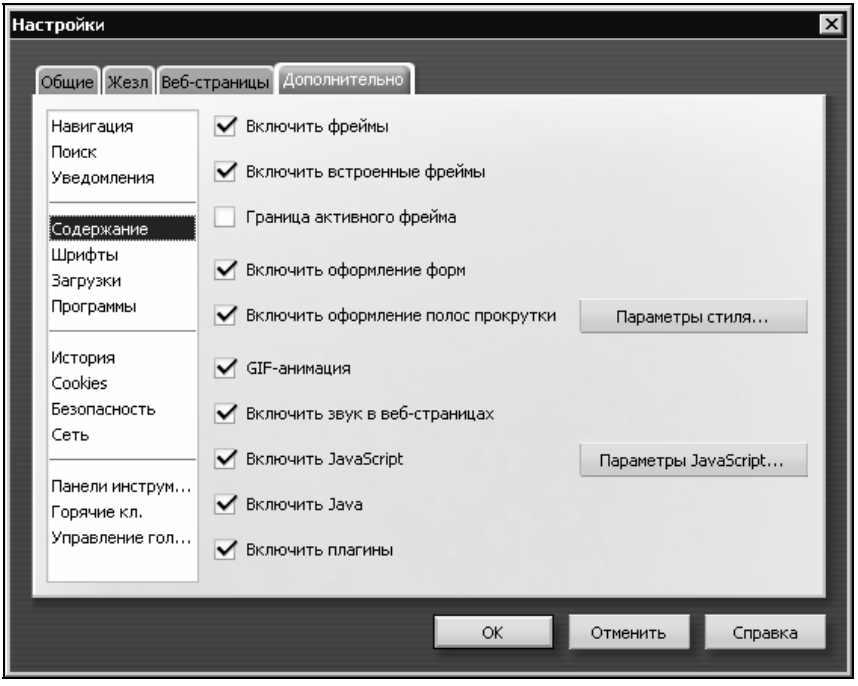


Рис. 16.3. Диалоговое окно **Настройки** Opera (вкладка **Дополнительно**)

ки, включить флажок **Открывать консоль JavaScript при ошибке** в появившемся на экране диалоговом окне **Параметры JavaScript** (рис. 16.4) и последовательно нажать кнопки **OK** окон **Параметры JavaScript** и **Настройки**.

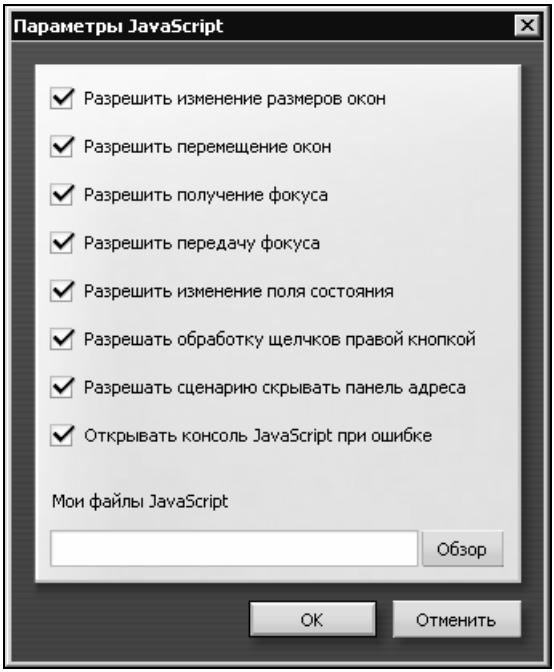


Рис. 16.4. Диалоговое окно **Параметры JavaScript** Opera

Если Opera встретит в Web-сценарии ошибку, она выведет на экран окно *консоли JavaScript* (рис. 16.5). Бóльшую часть этого окна занимает список ошибок, встретившихся в коде сценария; каждый пункт списка представляет собой развернутое описание ошибки, включая ее характеристику и местоположение в коде сценария.

Отметим, что мы можем менять размеры окна консоли JavaScript, делать его неактивным, переключаясь в окно самого Web-обозревателя, и минимизировать его. Зачастую это очень удобно.

Кроме списка ошибок, окно консоли JavaScript содержит три кнопки:

- **Очистить** — очищает список ошибок;
- **Свернуть** — минимизирует окно консоли JavaScript;
- **Закрыть** — закрывает это окно.

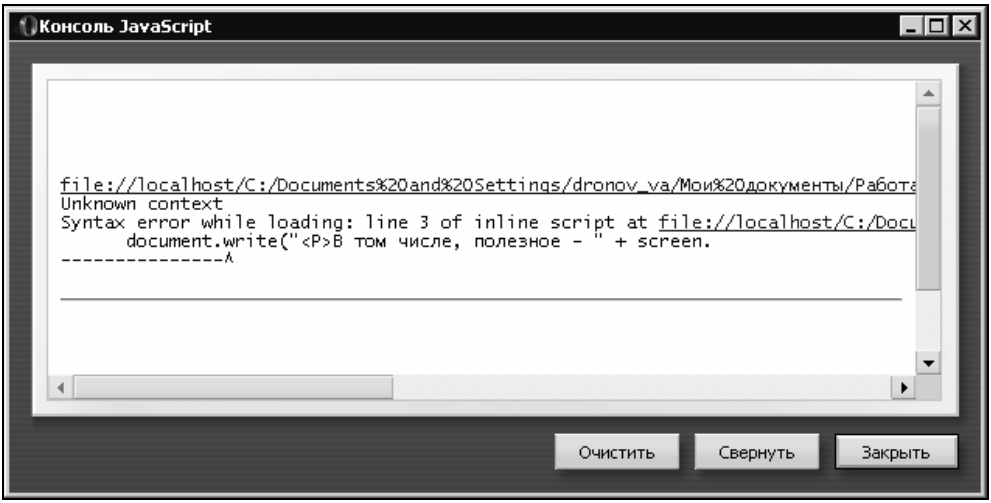


Рис. 16.5. Окно консоли JavaScript Opera

Народ советует

При отладке сложных сценариев, закончив очередной этап работы, следует очищать список ошибок консоли JavaScript. Иначе он скоро переполнится, и нам будет трудно в нем ориентироваться.

Содержимое списка консоли JavaScript не сохраняется после закрытия Opera. Так что мы не сможем отложить работу над ошибками на потом.

Кроме того, мы можем вывести окно консоли JavaScript вручную. Для этого достаточно выбрать пункт **Консоль JavaScript** подменю **Дополнительно** меню **Инструменты**.

Решение (Firefox)

Firefox — программа "скрытная". Никаких предупреждений об ошибках она сама не выдает. Нам придется вручную вывести на экран консоль JavaScript и посмотреть, что же Firefox не понравилось в нашем коде.

Вывести консоль JavaScript Firefox очень просто. Выбираем пункт **Консоль JavaScript** меню **Инструменты** — и окно консоли на экране (рис. 16.6). Как им пользоваться, мы уже знаем по Opera.

В панели инструментов в верхней части окна консоли JavaScript присутствует набор кнопок. Давайте его рассмотрим.

Четыре кнопки, объединенные в группу, позволят нам просмотреть в списке консоли только нужные нам пункты. Вот эти кнопки:

- **Ошибки** — включает вывод в списке только *критических ошибок* (приводящих к прерыванию выполнения сценария);

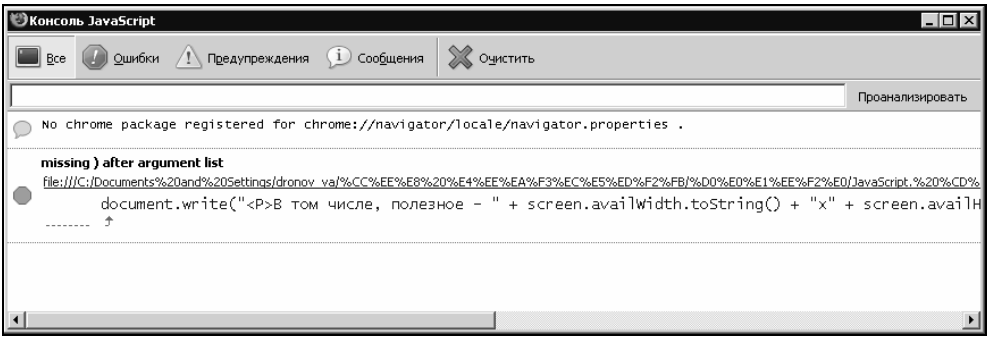


Рис. 16.6. Окно консоли JavaScript Firefox

- **Предупреждения** — включает вывод в списке предупреждений о несоблюдении стандартов в коде JavaScript (например, Firefox очень "не любит", когда в сценариях используется коллекция `all` объекта `document` — вместо нее предлагается использовать метод `getElementById` этого же объекта);
- **Сообщения** — включает вывод в списке информационных сообщений от самой программы Firefox;
- **Все** — включает вывод в списке и критических ошибок, и предупреждений, и сообщений.

Последняя кнопка — **Очистить** — вызывает очистку списка консоли JavaScript. При этом удаляются и критические ошибки, и предупреждения, и сообщения.

Несмотря на свою, скажем так, ненавязчивость в смысле предупреждения об ошибках, Firefox, тем не менее, предлагает Web-программистам еще один весьма мощный и полезный инструмент для исследования структуры Web-страницы. Это *инспектор DOM*. Вызвать его можно выбором пункта **Инспектор DOM** меню **Инструменты**, после чего на экране появится его окно (рис. 16.7).

Окно инспектора DOM разделено на две части. Мы можем перетаскивать мышью разделяющую их серую полосу, меняя относительные размеры этих частей. А сейчас давайте их рассмотрим.

Итак, в левой части окна находится иерархический список, представляющий собственно структуру Web-страницы. Пункты этого списка делятся на две группы: теги и текстовые элементы. Пункты-теги представлены теми же именами, что и теги, которые они представляют (например, пункты **HEAD** и **SCRIPT** на рис. 16.7 представляют одноименные теги). Пункты — текстовые элементы представлены именами **#text**. К текстовым элементам относятся любые фрагменты текста, находящиеся внутри тегов, — текст абзацев, заголовков, код Web-сценариев и пр.

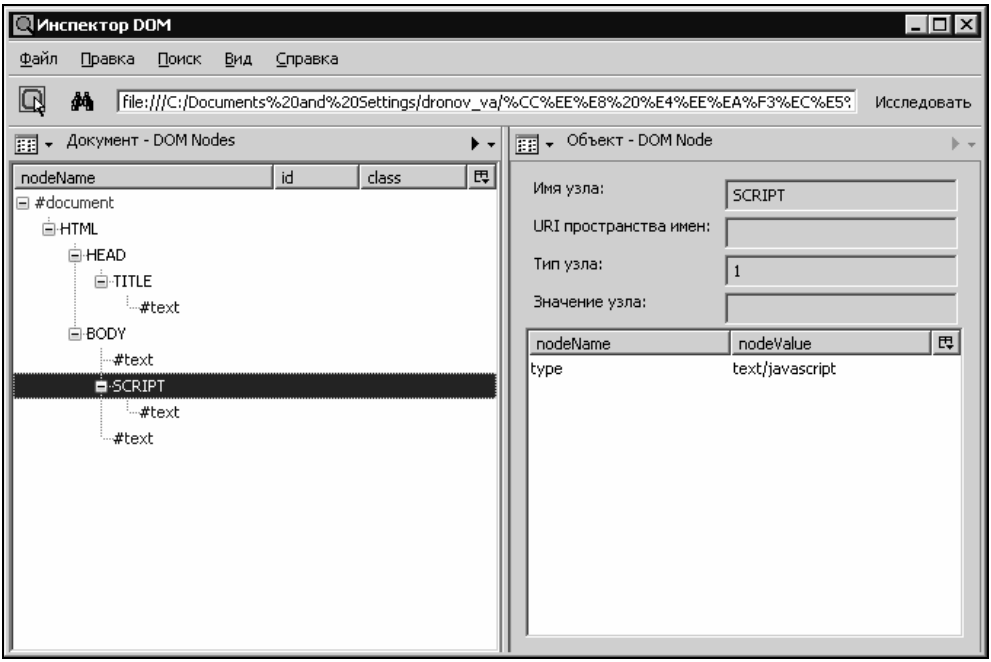


Рис. 16.7. Инспектор DOM Firefox (отображены сведения о теге <SCRIPT>)

В правой же части окна инспектора DOM помещается большая панель просмотра, отображающая сведения о выбранном в иерархическом списке слева пункте. Это могут быть как краткие сведения о выбранном теге (как на рис. 16.7), так и содержимое текстового элемента (см. рис. 16.8 — там представлен код Web-сценария, помещенного в тег <SCRIPT>).

Вообще, у расположенной в правой части окна панели просмотра несколько режимов работы. Для переключения между этими режимами служит кнопка, расположенная у верхнего конца разделяющей левую и правую части окна серой полосы и показанная на рис. 16.9. Если нажать эту кнопку, на экране под ней появится небольшое меню, поэтому такие кнопки и называются *кнопками с меню*.

Тот режим работы панели просмотра, что мы только что изучили, включается выбором пункта **DOM Node** этой кнопки с меню. Если же выбрать пункт **Javascript Object**, панель просмотра будет выглядеть так, как показано на рис. 16.10. Прямо в ней появится иерархический список, представляющий все свойства экземпляра объекта, соответствующего выбранному в левом списке тегу или текстовому элементу, и их текущие значения. Очень полезная штука!

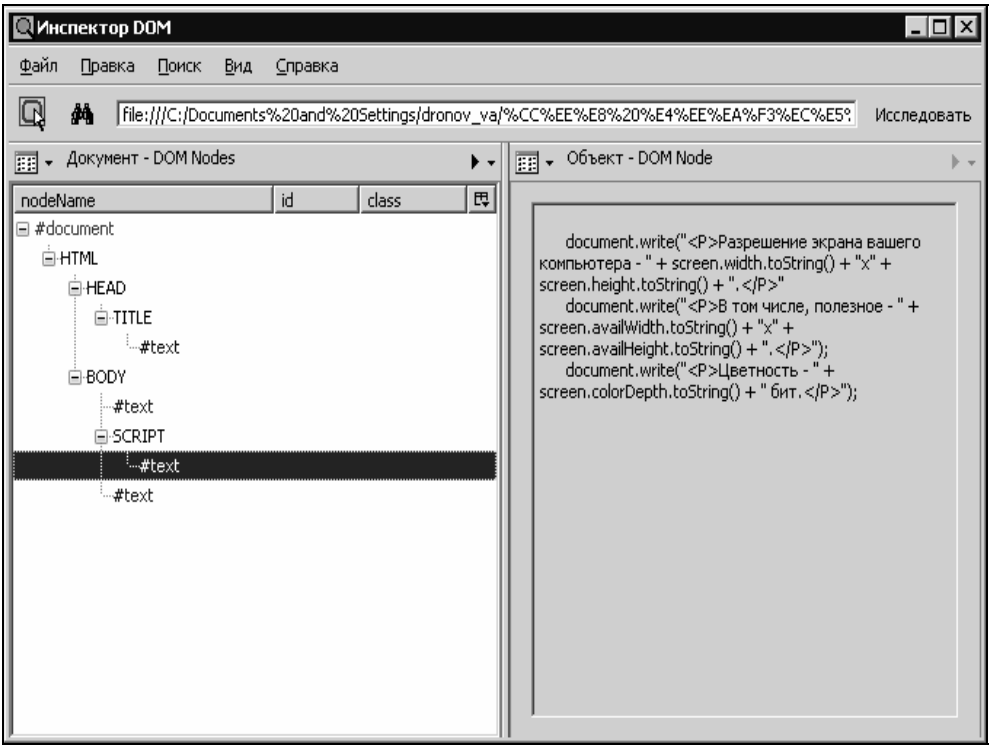


Рис. 16.8. Инспектор DOM Firefox
(отображено содержимое текстового элемента — содержимое сценария)

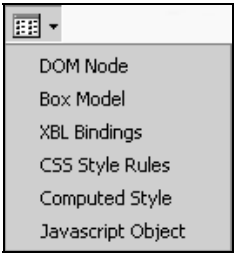


Рис. 16.9. Кнопка переключения режимов работы панели просмотра
в инспекторе DOM Firefox с открытым меню

Пожалуй, Firefox действительно предлагает Web-программисту больше, чем другие Web-обозреватели. Даже намного больше.

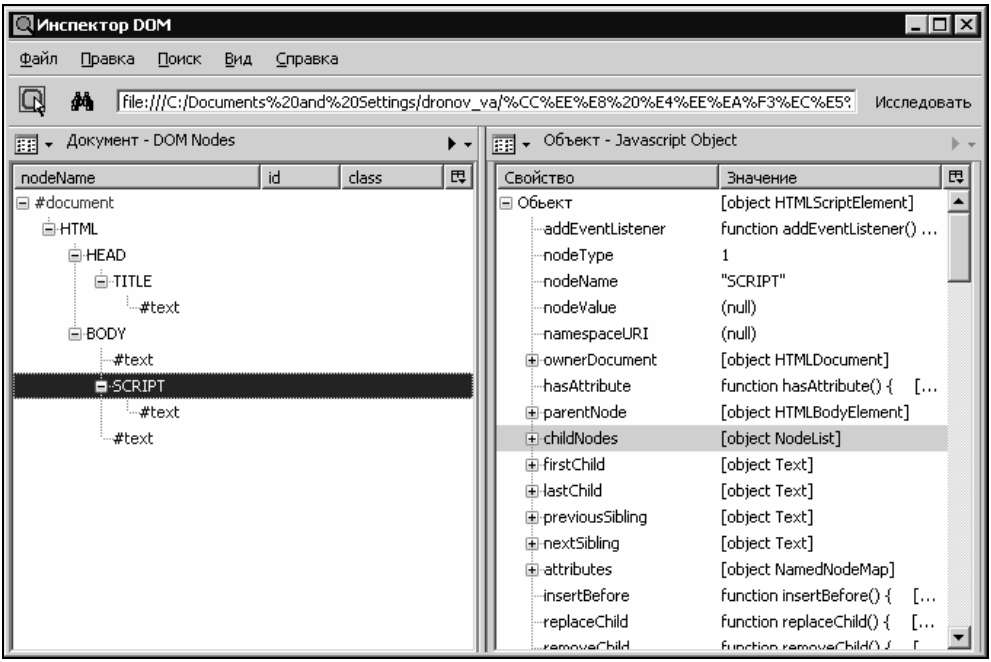


Рис. 16.10. Инспектор DOM Firefox (отображены сведения об экземпляре объекта, соответствующем тегу <SCRIPT>)

Как проследить выполнение сценария?

Проблема

Мне нужно проследить за процессом выполнения Web-сценария, иначе говоря, выполнить его *трассировку*. Есть ли в существующих Web-обозревателях какие-то средства для этого?

Решение 1

Нет — увы! Но можно использовать для трассировки средства самого JavaScript, изученные нами в *главе 12*. А именно — окна-сообщения, выводимые методом `alert` объекта `window`.

Чтобы выяснить, выполняется ли какой-либо фрагмент кода, нам достаточно будет вставить в него выражение, вызывающее вышеупомянутый метод. В качестве параметра этого метода (текста, выводимого в окне-сообщении) можно использовать что-то вроде "!!!" или "Код выполнен!".

```
// Какой-то код
window.alert("!!!");
// Какой-то код
```

Точно таким же образом можно узнать, какое значение принимает какая-либо переменная в данный момент. Для этого достаточно передать эту переменную методу `alert`:

```
// Код, вычисляющий значение нужной переменной
window.alert(<Нужная переменная>);
```

Достоинство этого подхода в том, что он очень прост. Недостаток же — необходимость каждый раз для продолжения работы сценария закрывать появившееся окно-сообщение нажатием кнопки **OK**, что не всегда приемлемо, но всегда быстро надоедает.

Решение 2

Можно выполнить вывод каких-то сообщений в отдельное окно Web-обозревателя, открытое программно. Для этого в самом начале сценария помещаем вызов описанного в *главе 4* метода `open` объекта `window`, открывающего это окно:

```
var traceWindow = window.open("", "trace");
```

Первым параметром мы передаем этому методу пустую строку, т. к. нам нужно открыть пустое окно, не содержащее никакой Web-страницы. Вторым параметром мы передаем имя создаваемого окна (в нашем случае — `trace`). Третий параметр, задающий параметры создаваемого окна, можно в нашем случае опустить.

После этого в нужном месте кода мы просто вписываем такое выражение:

```
traceWindow.document.write(<Выводимый текст в строковом виде>);
```

Лучше всего поместить выводимый текст в тег `<P>`, иначе весь вывод в созданном ранее окне будет выполняться в одну строку. Например:

```
traceWindow.document.write("<P>Выражение выполнено!</P>");
```

или

```
traceWindow.document.write("<P>Переменная a = " + a.toString() + "</P>");
```

Достоинство этого подхода в том, что трассировка не прерывает выполнения сценария. Серьезных же недостатков у него не замечено.

Решение 3

Использовать для вывода сообщений в отдельное окно объект `Trace` (листинг 16.1), точнее, его единственный экземпляр `jspsTrace`, создаваемый самим сценарием, приведенным в этом листинге. Это позволит нам сразу же начать трассировку, не отвлекаясь на создание экземпляра объекта `Trace`.

Объект `Trace` поддерживает два метода — основной и вспомогательный. Основной — это метод `write`, формат вызова которого таков:

```
write(<Выводимый текст>);
```

Единственным параметром этому методу передается выводимый в создаваемое окно текст. Он должен быть представлен в любом виде, не только строковом, — преобразование его к строковому выполнит сам объект. Также сам объект поместит выводимое значение внутрь тега `<P>`. Результата этот метод не возвращает.

Вспомогательный метод — это `close`. Он закрывает созданное для вывода трассировочных сообщений окно. Параметров он не принимает и результата не возвращает.

Листинг 16.1. Объект `Trace` — простейший трассировщик

```
function Trace()
  {
    this.window = null;
    this.write = mjspsTWrite;
    this.close = mjspsTClose;
  }

function mjspsTWrite(pOutput)
  {
    if ((!this.window) || (this.window.closed))
      this.window = window.open("", "trace");
    this.window.document.write("<P>" + pOutput.toString() + "</P>");
  }

function mjspsTClose()
  {
    this.window.close();
    this.window = null;
  }

var jspsTrace = new Trace();
```

Хорошая идея!

Поместите объявление этого объекта в файл сценариев trace.js. Впоследствии, чтобы использовать его, достаточно будет просто подключить к Web-странице этот файл сценариев с помощью тега:

```
<SCRIPT SRC="trace.js"></SCRIPT>
```

Пример

Предположим, мы имеем Web-страницу, выводящую с помощью простого сценария сведения о видеоподсистеме компьютера посетителя. HTML-код этой страницы приведен далее.

```
<HTML>
  <HEAD>
    <TITLE>Параметры видеоподсистемы</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      document.write("<P>Разрешение экрана вашего компьютера - " +
      ↳screen.width.toString() + "x" + screen.height.toString() +
      ↳".</P>");
      document.write("<P>В том числе, полезное - " +
      ↳screen.availWidth.toString() + "x" + screen.availHeight.toString()
      ↳+ ".</P>");
      document.write("<P>Цветность - " + screen.colorDepth.toString() +
      ↳" бит.</P>");
    </SCRIPT>
  </BODY>
</HTML>
```

Теперь предположим, что мы хотим выполнить ее трассировку с помощью объекта `Trace`. В этом случае мы должны ее модифицировать следующим образом (вставленный код выделен полужирным шрифтом):

```
<HTML>
  <HEAD>
    <TITLE>Параметры видеоподсистемы</TITLE>
    <SCRIPT SRC="trace.js"></SCRIPT>
    <!-- Предполагается, что объявление объекта Trace было помещено
    в файл сценариев trace.js -->
  </HEAD>
  <BODY>
    <SCRIPT TYPE="text/javascript">
      document.write("<P>Разрешение экрана вашего компьютера - " +
      ↳screen.width.toString() + "x" + screen.height.toString() +
      ↳".</P>");
      jspsTrace.write("Первое выражение выполнено!");
      document.write("<P>В том числе, полезное - " +
      ↳screen.availWidth.toString() + "x" + screen.availHeight.toString()
      ↳+ ".</P>");
```

```
      jspsTrace.write("Второе выражение выполнено!");
      document.write("<P>Цветность - " + screen.colorDepth.toString() +
      ↳" бит.</P>");
      jspsTrace.write("Третье выражение выполнено!");
      window.alert("Трассировка закончена.");
      jspsTrace.close();
    </SCRIPT>
  </BODY>
</HTML>
```

При открытии этой страницы на экране появится второе окно Web-обозревателя, созданное программно. В этом окне будут выведены три строки, сообщающие о выполнении первого, второго и третьего выражений сценария страницы. Напоследок на экране появится окно-сообщение, оповещающее об окончании трассировки, и после нажатия кнопки **OK** этого окна созданное программно окно-"трассировщик" будет закрыто.

Также обратим внимание, что нам не нужно создавать экземпляр объекта `Trace` самим, т. к. он уже создан в коде, помещенном в файл сценариев trace.js. Мелочь — а приятно!

Заключение

Что ж, книга закончилась. Народу больше нечего нам сказать.

Хотя, конечно, сказать есть что. Но всего этого не вместит никакая, даже самая толстая книга. Слишком сложен и многогранен JavaScript, слишком много у него областей применения. Слишком сложны и многогранны современные Web-обозреватели, особенно Internet Explorer, гораздый на сюрпризы для программистов. Слишком много кода на JavaScript написано за почти десятилетнюю историю существования этого языка, чтобы разобрать его работу. Слишком непредсказуема жизнь программиста, чтобы предсказать, что может пригодиться ему через неделю или даже через час.

И слишком быстро все меняется в интернет-технологиях...

Возьмем, например, самую "горячую" новинку сезона — технологию *AJAX* (Asynchronous JavaScript And XML, асинхронный JavaScript и XML). Это даже не технология, а набор подходов и приемов программирования, позволяющих создавать Web-страницы, выглядящие как настоящие Windows-приложения и работающие зачастую так же. Web-страница, построенная на основе AJAX, обменивается с серверной программой небольшими фрагментами данных в виде текста или XML и обновляет свое содержимое без перезагрузки. Сейчас на основе этой технологии создаются почтовые Web-серверы, электронные магазины, всевозможные корпоративные решения и пр.

А ведь AJAX — это тот же самый хорошо нам знакомый JavaScript! Да-да, именно стандартные возможности JavaScript и стандартные же возможности Web-обозревателей использует эта набирающая популярность технология.

Еще JavaScript используется для написания серверных Web-страниц (Web-страниц, содержащих включения серверных сценариев, выполняющихся серверной программой). Еще на нем пишут сценарии *WSH* (Windows Scripting Host), выполняющиеся на клиентских компьютерах, своего рода наследники

старых командных файлов MS-DOS. Еще JavaScript неплохо себя чувствует в качестве средства создания подключаемых модулей к различным программным пакетам (самый известный пример — Adobe Flash, ранее Macromedia Flash). Еще... да мало ли что!..

Скоро выйдет в свет новая операционная система Microsoft под названием Windows Vista. Она будет поддерживать выполнение так называемых гаджетов (gadgets) — небольших приложений, помещаемых на особую панель или прямо на рабочий стол. Эти гаджеты также будут создаваться на JavaScript (конечно с привлечением HTML, CSS, интернет-графики GIF, JPEG, PNG и Shockwave/Flash). Вот!

Так что без JavaScript никуда. И помимо Web-программирования у него найдется работенка...

А в помощь начинающим Web-программистам приведем список интернет-ресурсов, содержащих полезные для них сведения:

- **http://msdn.microsoft.com/ie/** — "домашний" сайт Internet Explorer. Исчерпывающее описание Web-обозревателя фирмы Microsoft, HTML, CSS, JavaScript и многого другого;
- **http://www.w3c.org** — сайт самого великого и ужасного комитета W3C, занимающегося интернет-стандартами;
- **http://www.webscript.ru/** — сайт, посвященный Web-программированию;
- **http://www.webmascon.com/** — электронный журнал для Web-дизайнеров. Будет полезен и Web-программистам;
- **http://docs.rinet.ru/** — огромное количество документации для программистов, и не только по HTML и JavaScript. Правда, по-английски и зачастую устаревшей;
- **http://www.sources.ru/** — архив бесплатных исходных текстов программ. Есть и сценарии JavaScript;
- **http://javascripts.boom.ru/** — архив бесплатных сценариев JavaScript;
- **http://dynapi.sourceforge.net/** — "домашний" сайт библиотеки DynAPI — набора полезных сценариев JavaScript.

Ну, вот и все! Народ прощается с вами и желает успехов в программировании!

А вместе с народом прощается и автор этой книги —

Владимир Дронов

Список литературы

1. Вайк Аллен и др. JavaScript. Энциклопедия пользователя. Пер. с англ. — К.: ООО "Тидс ДС", 2001. — 480 с.
2. Дунаев В. Самоучитель JavaScript. 2-е издание. — СПб.: Питер, 2005. — 395 с.
3. Дронов В. А. JavaScript в Web-дизайне. — СПб.: БХВ-Петербург, 2001. — 800 с.

Предметный указатель

П

Р

С

Т

У

Ф

Х

Ц

Ч

Э

Я