JavaScript & DHTML Cookbook

Danny Goodman

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris - Sebastopol • Taipei • Tokyo

JavaScript и DHTM
Сборник рецепта
ДЛЯ ПРОФЕССИОНАЛОВ

Д. Гудма

ПИТЕР®

Москва • Санкт-Петербург • Нижний Новгород • Воронеж
Новосибирск • Ростов-на-Дону • Екатеринбург • Самара
Киев • Харьков • Минск
2004

ББК 32 988.02-018
УДК 681 324
Г93

Гудман Д.
Г93 JavaScript и DHTML. Сборник рецептов. Для профессионалов — СПб.: Питер, 2004.— 523 с: ил.

ISBN 5-94723-817-9

Эта книга посвящена совместному использованию HTML и JavaScript для создания интерактивных веб-страниц. На многочисленных практических примерах рассматриваются все возможности, которые язык JavaScript дает разработчику, от простых и очевидных до самых сложных.

Книга построена как сборник готовых к применению рецептов, которые будут полезны как начинающим разработчикам, знающим HTML и основы JavaScript, так и опытным специалистам, ищущим новые идеи и технологии. Автор отдает предпочтение новейшим стандартам, поэтому предлагаемые методики останутся актуальными в течение долгого времени.

Помимо решений задач, в книге имеется справочная информация, которая может потребоваться при адаптации рецептов к потребностям разработчика.

ББК 32.988.02-018
УДК 681 324

ISBN 0-596-00467-2 (англ.)
ISBN 5-94723-817-9

Краткое содержание

Содержание

Предисловие

Несколько закладок в моем браузере ссылаются на сайты, на которых годами висела табличка «сайт в стадии разработки» и черно-желтый знак «идет реконструкция». Я не забываю про них и периодически посещаю эти сайты, с оптимизмом ожидая тот день, когда исчезнет фигурка в каске, вооруженная лопатой, и появится блестящий сайт с долгожданной информацией.

Эта картина возникает передо мной, потому что я годами обдумывал данную книгу, но фигурка на знаке продолжала копать, пилить и стучать молотком, а книга все так же оставалась «в стадии разработки». Хотя клиентские Java-сценарии использовались с конца 1995 года, а концепции настоящего Dynamic HTML появились в 1998, это был нелегкий путь для тех, кто пытался использовать данные технологии (или писать о них). Противоречия в реализации объектной модели документа (DOM) в различных браузерах, осложненные развитием и изменением рекомендаций Word Wide Web Consortium (W3C), сделали развертывание масштабных клиентских приложений неожиданно сложным. Это особенно справедливо, если приложение должно работать на большинстве браузеров и платформ.

Наконец, после того как был реализован стандарт W3C DOM, а другие стандарты, относящиеся к DHTML (HTML, XHTML и CSS), доработаны и приведены к единому виду, настало время применить это произведение искусства к типичным задачам для клиентских сценариев. Написанная в стиле сборника рецептов, эта книга должна дать ответы на самый распространенный вопрос у программистов, дизайнеров и авторов: «Как... ?»

О читателе

Клиентские сценарии и DHTML — настолько обширная область, что у читателей этой книги разные уровни знаний, ожидания и, возможно, опасения. Ни в одной книге нельзя даже надеяться охватить все вопросы, которые могут возникнуть у тех, кто использует DHTML-технологии. Поэтому мы позволили себе составить портреты читателей, имеющих разный опыт.

- У вас есть как минимум зачаточные знания о концепциях клиентских Java-сценариев. Вы умеете добавлять сценарии к страницам как с помощью тегов <script>, так и подключая внешний .js файл к странице. Вы также знаете, что такое переменные, строки, числа, булевские значения, массивы и объекты, пусть даже не зная точно, как использовать их в языке JavaScript. Эта книга —

не учебник, но читая вступление к каждой главе и последующее обсуждение, можно многому научиться.

- Путешествуя по Сети, вы могли встретить сделанный с помощью DHTML эффект, который хотели бы добавить к своей странице, но либо не можете расшифровать код, либо хотите создать свою собственную версию, чтобы избежать проблем с авторскими правами. Если нужный эффект или методика достаточно популярны, в этой книге найдется нужный рецепт. Можно использовать его таким, какой он есть, или изменить под свои цели, не беспокоясь об авторских правах и гонорарах, пока вы не предлагаете эти рецепты другим, как часть коллекции сценариев. Но, конечно, было бы замечательно, если вы упомяните эту книгу в комментариях к своему коду!
- Возможно, вы — опытный разработчик, который осторожно изучает создание клиентских сценариев. Из-за страшных историй о несовместимости браузеров вы занимались исключительно созданием серверных программ. Но теперь, когда многие модные сайты начали использовать клиентские сценарии для облегчения пользования, вы готовы взглянуть по-новому на эту технологию.
- Наконец, дальний край спектра: вы можете быть опытным DHTML-разработчиком, которому необходимы новые идеи и технологии. Например, вы вели разработки, ориентированные исключительно на Internet Explorer под Windows, но теперь стремитесь использовать стандартный синтаксис.
- Почти каждый читатель обнаружит, что некоторые рецепты в этой книге слишком просты, а другие — чрезмерно сложны для его уровня. Надеюсь, те, что посложнее, подвигнут вас к дальнейшему изучению предмета и улучшению своих навыков. Даже если вы все это знаете, удостоверьтесь, что не пропустили новые для вас советы в обсуждениях рецептов.

О рецептах

Читателю может быть полезно узнать, что автор имел в виду под названием книги. Чтобы продолжить метафору поваренной книги, я определенным образом оформил мой код и применил стили программирования, как шеф-повар оформляет свои приемы и применяет приправы.

Важен не только способ решения задач, но и ниточки стиля, которые вплетены в ткань кода. Поскольку приведенные примеры программ могут послужить основой для ваших собственных разработок, они написаны максимально ясно, чтобы упростить, как я надеюсь, понимание логики работы. Названия переменных, функций, объектов и прочих подобных вещей выбирались так, чтобы пояснять их назначение в коде. Одна из целей кодирования в том, чтобы каждая операция была бы очевидной для каждого, кто читает код, даже если это программист, который спустя полгода меняет код, чтобы исправить ошибку или добавить новую возможность. Нет смысла умничать, если никто не сможет понять, что имелось в виду, когда некоторое значение присваивалось переменной x.

В этой книге без тени смущения используется способ адресации объектов документа, предложенный в W3C DOM. Этот формат можно применять, чтобы

ссылаться на элементы документа в браузерах, начиная с Internet Explorer 5 и Netscape Navigator 6, то есть в большинстве браузеров, имеющихся на сегодняшний день. Там, где IE не поддерживает стандарт (например, в обработке событий), все рецепты содержат эффективные средства для того, чтобы код не зависел от браузера. Здесь вы не найдете много решений, ориентированных исключительно на IE, тем более — только на IE для Windows.

В то же время в книге признается, что вам, возможно, потребуется поддерживать пользователей, в браузерах которых не поддерживается W3C DOM. Большое внимание уделяется тому, чтобы DHTML-сценарии не мешали работе старых браузеров, которые их поддерживают. Это особенно важно, если страничка должна обслуживать пользователей с ограниченными возможностями (например, с ухудшенным зрением или физическими ограничениями) или карманные компьютеры с упрощенными браузерами.

Основное кредо всех рецептов в этой книге: сценарии должны повышать функциональность статичного содержания страницы. Не ищите здесь сценариев, которые будут менять цвет фона, раздражая пользователя, или заставлять элементы прыгать по всему экрану, распевая «С днем рожденья!» Прочитав эту книгу, вы сами сможете придумать, как сделать эти ужасные вещи, но это уже ваше личное дело. Приведенные здесь примеры, будучи немного консервативными, предполагают решение реальных задач, с которыми сталкиваются авторы и разработчики в профессиональных приложениях.

Техника написания и синтаксис в этой книге разработаны так, чтобы обеспечить максимальную совместимость в будущем. Будущее любой технологии сложно предсказуемо, но на сегодняшний день стандарты W3C DOM и ECMAScript, реализованные в новейших браузерах, являются наиболее стабильной платформой для построения клиентских приложений со времени появления этой технологии. С небольшими вставками кода для нормальной работы в старых браузерах программы должны работать достаточно хорошо и в будущем.

Содержание книги

В первых четырех главах уделяется внимание фундаментальным разделам JavaScript. В первой главе «Строки» показано различие между строчным значением и объектом String. Большую роль в анализе строк в рецептах играют регулярные выражения. Здесь также приведена универсальная библиотека для чтения и записи текста в файлы cookie. Вторая глава «Числа и даты» включает рецепты как по форматированию и преобразованию чисел, так и по вычислениям с датами, которые будут использованы в последующих рецептах. Глава 3 «Массивы и объекты» является, пожалуй, одной из важнейших глав, посвященных центральным понятиям языка JavaScript. Рецепты этой главы дадут ключи к созданию одномерных и многомерных массивов, их сортировке, созданию объектов, имитации хэш-таблиц и покажут возможности наследования. Глава 4 «Переменные, функции и управление последовательностью выполнения» содержит рецепты увеличения производительности сценария.

Главы с пятой по восьмую содержат решения задач, применимых практически ко всем браузерам с возможностью исполнения сценариев. Рецепты пятой

главы «Определение возможностей браузера» позволят освободиться от дурной привычки досконально проверять версию браузера, предлагая стандартные и совместимые методики для определения, возможно или нет выполнение данного блока кода. Если ваша цель — наплодить как можно больше окон, то глава 6 «Управление окнами» предлагает множество идей по организации их взаимодействия. Несколько рецептов предлагают способы организации модальных окон (или их подобия). Не каждый любит фреймы, но глава 7 «Управление фреймами» будет интересна всем, особенно тем, кто не хочет, чтобы его сайт был бы помещен во фрейм другого сайта. Тема главы 8 «Динамические формы» — интеллектуальные формы, движущая сила создания клиентских сценариев. Рецепты, переработанные с учетом современных технологий, включают в себя проверку данных формы (с использованием регулярных выражений и без них) и несколько интересных и тонких методик, использованных на некоторых популярных сайтах.

Взаимодействие с пользователем обеспечивается путем обработки событий, и в главе 9 «События» обсуждаются некоторые самые общие задачи, с которыми сталкиваются разработчики DHTML-сценариев. События важны для большинства рецептов оставшихся глав. Например, в главе 10 «Навигация по сайту», где вы увидите, как реализовать различные меню и передавать данные с одной страницы на другую. Глава 11 «Таблицы стилей» предлагает как простые, так и весьма прогрессивные рецепты, включая загрузку отдельной таблицы стилей в зависимости от браузера или операционной системы. Таблицам стилей много внимания уделяется в главе 12 «Визуальные эффекты для статичных элементов», изобилующей рецептами с интерактивными изображениями и настраиваемыми размерам шрифтов.

В главе 13 «Позиционирование элементов HTML» рассматриваются различные случаи, когда необходимо удерживать элементы страницы «на коротком поводке». Библиотека позиционирования, предложенная в одном из рецептов, широко используется по всей оставшейся части книги, включая рецепты для анимации элементов, прокручивания содержимого страницы и создания перетаскиваемого мышью элемента. Глава 14 «Динамическое содержимое» — хорошая проверка W3C DOM на таких задачах, как встраивание JavaScript и XML в документ, преобразование данных в страницу и сортировка HTML на стороне клиента. Несколько дополнительных, более сложных рецептов, посвященных созданию динамического содержимого, есть в главе 15 «Приложение DHTML» — это в том числе показ слайдов, графический таймер и всплывающий календарь для запроса даты у пользователя.

Браузеры

Каждый рецепт содержит оценку совместимости с браузерами. Эта оценка включает информацию о необходимой версии только для трех основных платформ: Internet Explorer для Windows, Internet Explorer для Macintosh и Netscape Navigator. В тех случаях, когда нет существенной разницы между версиями IE для Windows и Macintosh OS, подразумеваются браузеры под обе операционные системы. Число, записанное после названия браузера, означает младшую версию,

в которой будет работать описываемый рецепт. Секция обсуждения может содержать дополнительные варианты, работающие с более старыми или более новыми браузерами.

Оценка совместимости, показывающая Netscape 6 или выше, относится и к браузеру Mozilla, а также к большому числу браузеров, построенных на основе Mozilla (и ядре отображения HTML Gecko). Mozilla 1.0.1 и Netscape 7 аналогичны, так что рецепты, требующие Netscape 6, также подходят для Mozilla версии не меньше 1.0.

Прочие браузеры, такие как Opera и мириады других, с еще меньшим числом пользователей, не показаны в оценке совместимости. В некоторых дискуссиях особо затрагивается Opera, и по большей части рецепты из глав с первой по седьмую будут работать в Opera 5 или выше. Там же, где требуется более глубокое использование техник W3C DOM, в Opera зачастую возникают значительные проблемы. Opera 6 не обеспечивает достаточной поддержки W3C DOM для многих задач. Если же пользователи Opera составляют значительную часть от общего числа, то можно попробовать заставить сценарий работать в этом браузере или же считать его не исполняющим сценарии. Сценарии в этой книге рассчитаны на то, чтобы не мешать работе браузеров, не обрабатывающих сценарии, и вы можете использовать специальные средства, чтобы оградить пользователей Opera от некоторых более сложных сценариев.

Новый браузер от Apple, Safari, может оказать значительное влияние на рынок Macintosh, потеснив Internet Explorer с места доминирующего браузера для платформы Mac OS X. Но на время написания книги Safari, построенный на ядре KHTML, был в стадии бета-версии. Было бы неразумно пытаться обойти ошибки бета-версии и оценивать совместимость рецептов с незаконченным браузером. Остается надеяться, что производители новых систем будут соблюдать стандарты, разработанные их более опытными предшественниками, обеспечивая совместимость современных браузеров.

Может показаться, что внимание, уделяемое в последних главах WCD DOM, может исключить из рассмотрения множество старых браузеров (хотя библиотека DHTML из рецепта 13 совместима с браузерами, начиная с Netscape 4). Многие талантливые программисты продемонстрировали, что можно писать, обходя все ошибки интерпретаторов и заставлять приложение работать практически в любом браузере. Тем не менее, ради простоты кода, эта книга ориентирована на современные, наиболее популярные браузеры.

Оформление

Моноширинный шрифт использован для листингов HTML и сценариев.

Связь с издателями

Веб-страница, на которой можно загрузить примеры кода из этой книги, расположена по адресу http://www.oreilly.com/catalog/jvdhtmlckbk/.

Благодарности

Знак «идет реконструкция» был на этой книге с начала 1999 года, когда я согласился создать книгу рецептов DHTML. И хотя проект находился в этой стадии не так уж долго, это самый долгий срок, который пришлось выжидать, пока реальный мир и технологии не придут в соответствие с пониманием того, какой должна быть книга. Издательство было достаточно терпеливо, чтобы дать идее созреть, а также дождалось написания второй редакции книги DHTML: The Definitive Reference, и это напутствие всем издательствам, заботящимся не только о том, чтобы заполнить полки в книжных магазинах.

По счастливому стечению обстоятельств, незаконченная книга попала к Паоло Ферпосон (Paola Ferguson), одному из самых знающих и неутомимых редакторов, с кем мне довелось работать за свою 25-летнюю карьеру. Под ее редакцией вышли многие из наиболее успешных книг O'Reilly, она магически смешивала интересы автора, издательства и читателя, создавая конечный продукт.

Помогли и три дополнительные пары внимательных глаз, заглядывавших через плечо, чтобы дать критические замечания по техническим вопросам. В алфавитном порядке это Сью Гиллер (Sue Giller), Роберт Хастед (Robert Husted) и Дори Смит (Dori Smith). Они дали бесчисленное множество советов, которые помогли написать книгу и сценарии. Тройное им спасибо!

Выбирать для книги те или иные рецепты во многом помогала занимающаяся сценариями общественность. Прочитав тысячи обсуждений на сетевых форумах с 1996 года, выслушав читателей книги DHTML: The Definitive Reference и просмотрев поисковые запросы, приводившие пользователей на мой сайт (http://www.dannyng.com), я решил, что сумел выделить суть потребностей большей части разработчиков клиентских сценариев. Читатели, ваши страдания, смущение и растерянность не прошли незамеченными. Надеюсь, эта книга поможет вам и позволит ответить на все вопросы.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Все исходные тексты, приведенные в книге, вы можете найти по адресу http://www.piter.com/download.

На веб-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

1 Строки

1.0. Вступление

Строка — один из основных строительных блоков, с которыми работает JavaScript. Любой сценарий, соприкасающийся с URL или данными, которые пользователь вводит в поля формы, работает со строками. Большинство свойств объектной модели документа — строки. Данные, которые читаются или пишутся в файлы cookie, тоже строки. Строки используются буквально повсюду!

В ядре языка JavaScript имеется набор свойств и методов для работы со строками, которые можно найти в большинстве языков программирования. С их помощью можно разбирать строку символ за символом, если это нужно, менять регистр всех букв в строке или работать с отдельными ее частями. Большинство современных браузеров используют мощь регулярных выражений, которые значительно упрощают многие задачи по обработке строк, — стоит лишь пройти краткий курс обучения.

Сценарии обычно управляются значениями, имеющими строковый тип. Например, если вам необходимо получить текст, который пользователь ввел в поле ввода на форме, свойство value возвращает значение, уже имеющее строковый тип. Все свойства и методы работы со строками сразу доступны в сценарии для обработки значения такого поля ввода.

Создание строки

Есть несколько способов создать строку. Простейший из них — просто присвоить строку в кавычках переменной (или свойству объекта):

```
var myString="Fluffy is pretty cat "
```

Кавычки вокруг строки в JavaScript могут быть как одинарными, так и двойными, главное, чтобы они были одинакового типа. Например, допустимы оба показанных ниже выражения:

```
var MyString="Fluffy is a pretty cat.":
var myString='Fluffy is a pretty cat.'
```

Но следующая несовпадающая пара кавычек недопустима и приводит к ошибке сценария:

```
var myString="Fluffy is a pretty cat.';
```

Наличие двух наборов символов кавычек удобно, когда вам необходимо встроить одну строку внутрь другой. Показанный далее оператор document.write(), который должен выполняться при загрузке страницы, имеет одну наружную строку (это вся строка, которая будет выведена методом), содержащую в себе кавычки. Они ограничивают значения строчных атрибутов элемента HTML документа:

```
document.write("<img src='img/logo.jpg' height='30' width='100' alt='Logo'>");
```

Вы также можете менять на свой вкус порядок следования одинарных и двойных кавычек. Таким образом, показанный выше оператор будет выполнен так же, если его записать следующим образом:

```
document.write('<img src="img/logo.jpg" height="30" width="100" alt="Logo">');
```

Можно использовать и более двух уровней вложения, если применять escape-последовательности. В рецепте 1.8 показан пример использования escape-последовательностей.

Говоря техническим языком, описываемые строки не являются в строгом понимании *объектами* JavaScript. Они представляют собой строковые *значения,* которые могут использовать все свойства и методы глобального объекта типа String, наследуемого всеми окнами браузера, поддерживающего сценарии. Для всех операций по обработке строк в JavaScript желательно использовать именно строковые значения. Тем не менее иногда строчные значения JavaScript не совсем подходят. Вы можете столкнуться с такой ситуацией, когда используете сценарий для взаимодействия с Java-апплетом, одному из публичных методов которого требуется аргумент строчного типа. В этом случае может возникнуть необходимость создать полноценный объект типа String и передать его в качестве аргумента метода. Чтобы создать такой объект, используйте конструктор объекта String:

```
var myString=new String("Fluffy is a pretty cat.");
```

После того как этот оператор будет выполнен, переменная myString станет объектом, а не строкой. Но этот объект наследует все свойства и методы объекта String и может использоваться в работе с Java-апплетом.

Регулярные выражения

Непосвященному регулярные выражения могут показаться таинственными и сложными. Эту книгу нельзя назвать учебником по регулярным выражениям, но, возможно, рецепты этой главы будут вам интересны и подтолкнут к дальнейшему их изучению.

Назначение регулярных выражений в том, чтобы задать образец, с которым можно сравнивать строки. Если символы в строке соответствуют образцу, регулярное выражение может показать, где именно произошло совпадение, тем самым облегчая дальнейшую обработку строки (возможно, операцию поиска и замены). Регулярные выражения позволяют не только задавать простой образец из фиксированных символов. Например, вы можете задать образец как пята

цифр, ограниченных пробелами с каждой стороны. Другой образец может определять формат типичного адреса электронной почты, независимо от длины имен пользователя и домена, учитывая, что полное имя домена включает по крайней мере одну точку.

Таинственная часть регулярных выражений — это нотация, которая используется для того, чтобы задать различные условия в образце. Нотация регулярных выражений в JavaScript практически идентична нотации, используемой в таких языках, как Perl. Синтаксис совпадает практически везде, кроме некоторых загадочных случаев. Одно из принципиальных отличий состоит в том, как по образцу создается объект регулярного выражения. Вы можете использовать как формальный синтаксис с использованием конструктора, так и сокращенный. Два показанных ниже примера создают одинаковые объекты:

```
var re = /pattern/ [g j i j gi]:                    // Сокращенный синтаксис
var re = new RegExp(["pattern".["g "j "i" | "gi"]]): // Формальный конструктор
```

Необязательные завершающие символы определяют, будет ли регулярное выражение применено глобально, ко всем совпадениям, и чувствительно ли оно к регистру. Internet Explorer версии 5.5 и выше, а также Netscape версии 6 и выше воспринимают дополнительный модификатор m, который влияет на определение границ строк в многострочном тексте.

Для тех, кто уже знаком с регулярными выражениями, в табл. 1.1 описана нотация регулярных выражений, используемая в браузерах с NN 4 и IE 4.

Таблица 1.1. Запись регулярных выражений

Символ	Совпадает с	Пример
\b	Граница слова	/\bto/ совпадает с «tomorrow»
		/to\b/ совпадает с «Soweto»
		/\bto\b/ совпадает с «to»
\B	Все, кроме границы слова	/\Bto/ совпадает с «stool» и «Soweto»
		/to\B/ совпадает с «stool» и «tomorrow»
		/\Bto\B/ совпадает с «stool»
\d	Цифры от 0 до 9	/\d\d/ совпадает с «42»
\D	Не цифры	/\D\D/ совпадает с «to»
\s	Одиночный пробел	/under\sdog/ совпадает с «under dog»
\S	Не пробел	/under\Sdog/ совпадает с «under-dog»
\w	Цифра, буква или знак подчеркивания	/l\w/ совпадает с «1A»
\W	Не цифра, не буква и не знак подчеркивания	/1\W/ совпадает с «1%»
.	Любой символ, кроме символа новой строки	/../ совпадает с «Z3»
[...]	Любой символ из перечисленных в скобках	/J[aeiou]y/ совпадает с «Joy»

Символ	Совпадает с	Пример
[^...]	Любой символ, кроме перечисленных в скобках	/J[^eiou]y/ совпадает с «Jay»
*	Повторение ноль или более раз	/\d*/ совпадает с ««, «5», «444»
?	Ноль или один раз	/\d?/ совпадает с ««, «5»
+	Повторение один или более раз	/d+/ совпадает с «5», «444»
{n}	Повторение точно n раз	/\d{2}/ совпадает с «55»
{n,}	Повторение n или более раз	/\d{2,}/ совпадает с «555»
{n,m}	Повторение от n до m раз	/\d{2,4}/ совпадает с «55555»
^	Начало строки или всего текста	/^Sally/ совпадает с «Sally says ...»
$	Конец строки или всего текста	/Sally$/ совпадает с «Hi, Sally»

В рецептах 1.5-1.7, а также в рецепте 8.2 показано, как регулярные выражения могут облегчить различные операции проверки строк, обеспечивая меньший расход ресурсов по сравнению с традиционными процедурами работы со строками. Для более глубокого изучения регулярных выражений обратитесь ко второму изданию книги Дж. Фридла «Регулярные выражения» (Питер, 2003).

1.1. Конкатенация (объединение) строк

NN2 IE3

Задача

Необходимо слить две строки в одну или собрать одну строку из большого числа фрагментов.

Решение

Используя оператор сложения (+), можно объединять несколько строчных значений одним оператором:

```
var longString = "Один кусочек " + " плюс еще один кусочек";
```

Чтобы собрать строку из нескольких фрагментов в нескольких операторах, используйте оператор присваивания со сложением:

```
var result = "";
result += "Меня зовут " + document.myForm.myName.value;
result += "и мой возраст - " + document.myForm.myAge.value
```

Этот оператор обладает обратной совместимостью и имеет более компактную запись, чем менее элегантный подход:

```
result = result + "Меня зовут " + document.myForm.myName.value;
```

Обсуждение

Можно использовать несколько операторов конкатенации строк в одном выражении, но необходимо быть осторожным с переносами строк в коде. Интерпретаторы JavaScript автоматически вставляют точку с запятой в логических концах строк кода, поэтому нельзя просто разорвать строку, не окончив ее синтаксически корректно. Например, следующее выражение в виде, в котором оно записано, приведет к синтаксической ошибке при загрузке страницы:

```
var longString = "Один кусочек " + " плюс еще
один кусочек":
```

Интерпретатор воспринимает первую строчку кода так:

```
var longString = "Один кусочек " + " плюс еще;
```

С его точки зрения, это выражение содержит незаконченную строку, что приводит к ошибке как в этой строке, так и в следующей. Чтобы корректно разорвать строку кода, необходимо поставить закрывающую кавычку и знак сложения в конце строки кода (точку с запятой ставить не нужно, так как выражение еще не закончено). Также следует убедиться, что следующая строчка начинается с кавычки:

```
var longString = "Один кусочек " + " плюс еще " +
"один кусочек":
```

Также, для удобства чтения, можно вставлять в код пробелы, используя то, что пробелы вне кавычек игнорируются:

```
var longString = "Один кусочек " + " плюс еще   +
    "один кусочек":
```

Разрывы строк в коде не оказывают влияния на значение строк. Поэтому, если необходимо создать строчное значение, содержащее перенос строки, следует включить в строку одну из специальных escape-последовательностей (например, \п). Для оформления текста для окна сообщения с уведомлением, так чтобы возникла иллюзия двух абзацев, включите пару специальных символов перехода на новую строку:

```
var confirmString = "Вы не дали ответ на последний ответ." +
    "\п\пВы все равно хотите отправить форму ?":
```

Обратите внимание, что такой способ вставить разрыв строки подходит для диалоговых окон и прочих чисто текстовых контейнеров. Если интерпретировать такую строку, как HTML, то никаких переносов строк вставлено не будет. В этом случае необходимо явно включить тег
 в строку:

```
var htmlString = "Первая строчка текста<br>Вторая строчка текста".
```

Смотрите также

В рецепте 1.8 показано, как включать в строковое значение специальные управляющие символы, такие как возврат каретки.

1.2. Работа с фрагментами строк

NN2 IE3

Задача

Необходимо выделить часть строки.

Решение

Для того чтобы получить копию участка строки, начинающегося в указанном месте и заканчивающегося либо в конце строки, либо в другом указанном месте, необходимо использовать метод substring(), доступный во всех браузерах. При этом позиции для этого метода указываются от начала строки:

```
var myString = "Every good boy does fine.";
var section = myString.substring(0, 10);  // section равен "Every good"
```

Метод slice() аналогичен, но позиция конца фрагмента в нем отсчитывается от конца исходной строки (это отрицательное значение):

```
var myString = "Every good boy does fine.";
var section = myString.slice(11, 6);    // section равен "boy does"
```

Нестандартный, но поддерживаемый многими браузерами метод substr() получает в качестве второго параметра длину подстроки:

```
var myString = "Every good boy does fine.";
var section = myString.substr(6, 4);    // section равен "good"
```

Если фрагмент указанной длины выходит за границу строки, то возвращается участок до конца строки.

Обсуждение

Параметры для методов slice() и substring(), совместимых со стандартом ECMA, представляют собой целочисленные индексы, отсчитываемые с нуля. Первый параметр, указывающий начало фрагмента. обязателен. Если методу slice() передать два положительных аргумента, из которых второй больше первого, метод возвратит то же значение, что и substring() с теми же аргументами.

Необходимо отметить, что целочисленные аргументы, передаваемые методам substring() и slice(), действуют так, как будто указывают между отдельными символами строки. Следовательно, если передать методу substring() аргументы 0 и 4, будет выделена подстрока, включающая позиции с нулевой по первую, как показано на рис. 1.1.

Если в переданных методу substring() или substr() аргументах первый будет больше второго, интерпретатор JavaScript автоматически поменяет их. Метод slice() в подобном случае не столь простителен к ошибкам и возвращает пустую строку.

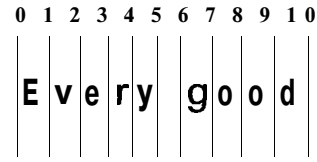


Рис. 1.1. Определение размера подстроки

Ни один из методов для работы с подстроками не меняет исходную переменную, поэтому возвращенное значение необходимо для дальнейшей обработки поместить в переменную или передавать другому методу или функции в качестве аргумента.

Смотрите также

Рецепт 1.5 демонстрирует, как проверить строку на наличие в ней заданной подстроки.

1.3. Смена регистра

NN2 IE3

Задача

Необходимо преобразовать все символы строки в верхний или нижний регистр.

Решение

Для смены регистра объект String имеет два метода, toLowerCase() и toUpperCase():

```
var myString = "New York";
var lcString = myString.toLowerCase();
var ucString = myString.toUpperCase();
```

Оба метода возвращают измененное значение, не меняя исходную строку. Если нужно заменить исходную строку модифицированной, следует присвоить возвращаемое значение исходной переменной:

```
myString=myString.toLowerCase();
```

Конечно же, не нужно снова объявлять переменную командой var.

Обсуждение

Поскольку в JavaScript, как и практически во всех других языках, операции со строками чувствительны к регистру, часто преобразование регистра используют для сравнения строк, введенных пользователем с некоторыми значениями. Это может быть необходимо, потому что пользователь может менять регистр букв произвольным образом, что мешает сравнению (см. рецепт 1.4).

Другое применение функций преобразования регистра — подготовка введенных данных к передаче в базу данных, которой требуется текст в нижнем (или в верхнем) регистре. В этом случае преобразование можно выполнять как при отсылке данных, так и в процессе ввода. Ниже показан пример обработчика события onchange, преобразующего буквы в поле ввода в заглавные:

```
<input type="text" name="firstName" id="firstName" size="20' maxlength="25"
   onchange="this.value=this.value.toUpperCase();"/>
```

Преобразованное значение в данном примере просто записывается обратно в поле ввода.

Смотрите также

В рецепте 1.4 показано, как преобразование регистра букв облегчает выполнение важной задачи.

1.4. Проверка строк на равенство

NN2 IE3

Задача

Необходимо сравнить введенную пользователем строку с некоторой известной строкой.

Решение

Для сравнения можно преобразовать строку в верхний или нижний регистр и использовать оператор сравнения:

```
if (document.myForm.myTextBox.value.toLowerCase() — "new york") {
    // обработка введенного текста
}
```

При таком использовании преобразования регистра содержимое поля ввода не меняется. (В рецепте 1.3 показано, как изменить регистр текста в поле ввода.)

Обсуждение

В JavaScript есть два типа операторов сравнения. Полностью стандартный и совместимый оператор (==) производит преобразование типов, чтобы привести оба операнда к одному типу. Рассмотрим следующие присваивания:

```
van stringA = "My dog has fleas.":
var stringB = new String("My dog has fleas."):
```

Обе этих переменных содержат одну и ту же последовательность символов, но имеют различный тип данных. Первая переменная — строчное значение, в то время как вторая — объект String. Если сравнивать эти переменные с помощью

оператора (==), интерпретатор JavaScript будет пытаться производить различные преобразования, проверяя, равны ли полученные значения. В данном случае значения будут равны, и следующее выражение:

```
stringA == stringB
```

возвращает **true.**

Другой тип оператора сравнения, оператор строгого сравнения (===). не производит никаких преобразований. Поэтому следующее выражение будет равно **false,** потому что сравниваемые переменные имеют различный тип:

```
stringA === stringB
```

Если логика кода требует проверки строк на неравенство, можно использовать операторы проверки на неравенство (!=) и строгой проверки на неравенство (!==). Например, неправильно введенный текст можно обработать следующим образом:

```
if (document.myForm.myTextBox.value.toLowerCase() != "new york") {
    // Обработка неправильно введенного текста
}
```

Различия между строгой и нестрогой формой оператора проверки на неравенство те же, что между формами оператора сравнения. Например, чтобы проверить, равно ли содержимое поля ввода (строковое значение) некоторому числу, можно использовать оператор сравнения так:

```
if (document.myFor.myTextBox.value == someNumericVar) { ... }
```

Или же можно предварительно привести переменные к одному типу:

```
if (parseInt(myFor.myTextBox.value)== someNumericVar) { ... }
```

Если вы привыкли к языкам с более строгой типизацией, можете практиковать такие преобразования и в JavaScript, что иногда повышает удобочитаемость сценария.

Смотрите также

Рецепт 2.1 демонстрирует преобразования строк в числа и обратно. Рецепт 3.3 показывает преобразования строк и массивов. Рецепт 3.13 — преобразование произвольного объекта в строку.

L.5. Проверка на вхождение подстроки ›ез использования регулярных выражений

NN2 **IE3**

Задача

Необходимо выяснить, содержится ли одна строка в другой, не используя регулярных выражений.

Решение

Для поиска подстроки служит метод **indexOf()**, искомая подстрока передается ему в качестве аргумента. Если в строке имеется указанная подстрока, метод возвращает номер первого символа вхождения. Если же в строке нет указанной подстроки, метод возвращает – 1.

Следующая конструкция может быть использована, когда необходимо выполнить действия при отсутствии вхождения:

```
if (largeString.indexOf(shortString)) ==  1) {
    // Действия, выполняемые  если shortString не входит в largeString
}
```

Чтобы обработать ситуацию, когда подстрока содержится где-то в проверяемой строке, можно использовать следующую конструкцию:

```
if (largeString.indexOf(shortString)) ==  1) {
    // Действия, выполняемые, если shortStnng входит в largeStnng
}
```

Такая запись подходит в том случае, если необходимо определить только сам факт вхождения подстроки, не интересуясь ее положением.

Обсуждение

Целочисленное значение, возвращаемое методом **indexOf()**, может также быть полезно во многих ситуациях. Например, если обработчик события, вызываемый всеми элементами, должен обрабатывать только те из них, чьи идентификаторы начинаются с определенных символов. В таком случае можно проверять, равно ли возвращаемое значение нулю (оно должно указывать на первый символ строки):

```
function handlrClick(evt) {
    var evt = (evt) ? evt: ((window.event) ? window.event   null):
    if (evt) {
        var elem = (evt.target) ? evt.target .((evt.srcElem) ?
            evt.srcElement : null),
        if (elem && elem.id.indexOf("menuImg) == 0) {
            // Обработка события для элементов, чей
            // идентификатор начинается с menuImg
        }
    }
}
```

Нужно помнить о том, что, в случае когда имеется несколько вхождений заданной подстроки, метод **indexOf()** всегда возвращает указатель на начало первого вхождения. Для того чтобы определить другие, нужно использовать необязательный второй аргумент метода **indexOf()**, который содержит номер начальной позиции для поиска. Для подсчета всех вхождений можно воспользоваться следующим компактным кодом:

```
function countInstances(mainStr. srchStr) {
    var count = 0:
    var offset = 0:
    do {
        offset = mainStr.indexOf(srchStr. offset);
        count += (offset !=  1) ? 1 : 0:
    } while (offset++ !=  1)
    return count:
}
```

Тем не менее подсчет числа вхождений гораздо проще с использованием регулярных выражений.

Смотрите также

В рецепте 1.6 показано применение регулярных выражений для проверки вхождения подстроки.

1.6. Проверка на вхождение подстроки с использованием регулярных выражений

NN4 IE4

Задача

Необходимо с применением регулярных выражений определить, содержится ли заданная строка в другой.

Решение

Следует создать регулярное выражение (шаблон) с глобальным модификатором (д) по заданной подстроке, после чего передать его в метод match() для проверяемого строкового значения или объекта String:

```
var re = /строковая константа/g:
var result = longString.match(re):
```

Когда регулярное выражение создается с глобальным модификатором, метод match() возвращает массив, содержащий все найденные вхождения. Если не найдено ни одного вхождения, метод возвращает null.

Обсуждение

Для практического использования показанного механизма необходимо добавить некоторый дополнительный код. Если строка, которая ищется, хранится в переменной, использовать показанный выше синтаксис невозможно. В этом случае необходимо применить конструктор регулярного выражения:

```
var shortStr =  "Framistan 2000":
var re = new RegExp(shortStr, "g"):
var result = longString.match(re);
```

После вызова метода match() нужно исследовать массив, возвращенный методом:

```
If (result) {
    alert("Найдено " + result.length + " вхождений текста:   + result[0]):
} else {
    alert("Не найдено ни одного совпадения.");
}
```

Массив, возвращаемый методом match(), состоит из найденных строк. Если в качестве шаблона для поиска используется фиксированная строка, все его элементы одинаковы (это объясняет, почему в предыдущем примере для сообщения в диалоговом окне можно использовать первое найденное значение). Если же регулярное выражение содержит более сложный шаблон, найденные строки, совпадая с шаблоном, могут отличаться друг от друга.

Использование модификатора g позволяет найти все совпадения (а не только первое). При этом длина массива будет показывать количество найденных совпадений. Для простой проверки на вхождение можно опустить этот модификатор. В этом случае при наличии совпадений возвращаемый массив всегда будет иметь длину 1.

Смотрите также

Раздел «Регулярные выражения» во вступлении к этой главе. Рецепт 8.2 для примера использования регулярных выражений при проверке правильности данных в форме.

1.7. Поиск и замена строк

NN4 **IE4**

Задача

Необходимо произвести над заданной строкой операцию поиска и замены всех вхождений.

Решение

Самый простой путь — использовать регулярные выражения и метод replace() объекта String:

```
var re = /строковая константа/g:
var result = mainString.replace(re.replacementString);
```

Метод replace() не меняет содержимого исходной строки, потому возвращаемое значение необходимо сохранить для дальнейшего применения в программе. Если не произведено ни одной замены, метод возвращает исходную строку. Следует удостовериться, что при создании регулярного выражения был указан модификатор д, который заставляет метод replace() заменять все вхождения. Если этого не сделать, будет заменено только первое.

Обсуждение

Чтобы на практике использовать этот код, могут потребоваться некоторые добавления. Если строка для поиска хранится в переменной, применить показанный выше синтаксис невозможно. Вместо этого необходимо использовать конструктор регулярного выражения:

```
var searchStr = "F2":
var replaceStr = "Framistan 2000":
var re = new RegExpCsearchStr. "g"):
var result = longString.replace(re. replaceStr):
```

При работе с элементами, содержащими текст, такими как поля формы или элемент страницы, удобно сразу же помещать измененный текст в элемент. Например, чтобы поместить текст из поля ввода в нескольких местах элемента div, помеченных символами (ph), можно использовать метод replace() следующим образом:

```
var searchStr = "\\(ph\\)":
var re = new RegExpCsearchStr. "g"):
var replaceStr = document.myForm.myName.value;
var div = document.getElementById("boilerplate");
div.firstChild.nodeValue = div.firstChild.nodeValue.replace(re. replaceStr):
```

Поскольку скобки имеют специальное значение при использовании регулярных выражений, им должен предшествовать символ обратной косой черты (\). В свою очередь, при записи строки этот символ сам имеет специальное значение и должен быть предварен еще одной косой чертой.

Можно реализовать поиск и замену и без помощи регулярных выражений, хотя это довольно неуклюже. Методика включает в себя поиск вхождения заменяемой подстроки с помощью indexOf(), вырезание из строки найденного текста и вставку нового фрагмента. Повторяя эти три действия, можно заменить все вхождения одной строки на другую. Такое могло быть необходимо в некоторых старых версиях браузеров, но сейчас регулярные выражения есть во всех браузерах с поддержкой сценариев.

Смотрите также

Раздел «Регулярные выражения» во вступлении к этой главе; рецепт 14.14, демонстрирующий дополнительные возможности замены текста на странице в современных браузерах.

1.8. Использование специальных символов и escape-последовательностей

NN2 IE3

Задача

Необходимо поместить в строку некоторые специальные ASCII-символы, такие как символ табуляции, переноса строки, возврата каретки и т. д.

Решение

Чтобы записать такие символы, используйте escape-последовательности, показанные в табл. 1.2. Например, для обозначения апострофа используется **\'**:

```
var msg = "Welcome to Joe\'s dinner!":
```

Обсуждение

Ядро языка JavaScript, как и у большинства других языков, дает возможность записывать специальные символы. Под специальным символом здесь понимаются не буквенно-цифровые символы и не знаки препинания, а символы, имеющие специальное значение, обычно относящиеся к пробельным символам. Сейчас из них чаще всего используются символы табуляции, разрыва строки и возврата каретки.

Запись специальных символов начинается с обратной косой черты, за которой следует символ, представляющий нужный код. Например, табуляция записывается как \t, а разрыв строки как \n. Символ обратной косой черты дает интерпретатору указание рассматривать следующий символ как специальный. В табл. 1.2 показаны используемые escape-последовательности и их значение. Чтобы поместить такие символы в строку, достаточно ввести в ее код нужную последовательность:

```
var confirmString = "Вы не ввели ответ на последний " +
    "вопрос.\n\nВы все равно хотите отправить форму ?":
```

Если нужно использовать такие символы вместе с переменными, содержащими строковые значения, удостоверьтесь, что символы записаны в кавычках:

```
var myStr = lineText1 + "\n" + lineText2;
```

Специальные символы можно использовать для форматирования текста в простейших диалоговых окнах (см. методы **alert()**, confirm() и prompt()), а также в элементе формы textarea.

Обратите внимание, что для включения в строку символа обратной косой черты необходимо напечатать его в коде дважды. В противном случае он будет интерпретирован как начало escape-последовательности. Escape-последователь-

ности для кавычек можно использовать, чтобы добавить кавычку в строку (простая кавычка будет интерпретирована как конец строковой константы).

Таблица 1.2. Строчные escape-последовательности

Escape-последовательность	Описание
\b	Возврат на одну позицию
\t	Горизонтальная табуляция
\n	Разрыв строки (новая строка)
\v	Вертикальная табуляция
\f	Подача страницы
\r	Возврат каретки
\"	Двойные кавычки
\'	Одинарная кавычка
\\	Обратная косая черта

Следует учитывать некоторые нюансы при проверке текста в элементе формы textarea. Дело в том, что перенос строки, введенный пользователем, может кодироваться различным образом. Если пользователь нажимает Enter в поле textarea, IE для Windows вставляет два символа: сначала \r, затем \n. Но IE для Macintosh в этом случае вставляет только один \r. Netscape 6 и старше использует для этого символ \n. Поведение Navigator 4 зависит от операционной системы: в Windows добавляется \r\n, в Mac OS — \r, а в Unix — \n. Такое разнообразие делает распознавание введенных пользователем разрывов строк непростым.

Обратный путь — формирование в сценарии текста для поля ввода, гораздо проще, потому что современные браузеры воспринимают все эти варианты. Поэтому, если вставить в строку \r\n, \r или только \n, все браузеры воспримут это как разрыв строки и преобразуют в свое внутреннее представление.

Смотрите также

Рецепт 1.1, содержащий советы по конкатенации строк, относящиеся также и к escape-последовательностям.

1.9. Чтение и запись строк в файлы cookie

NN2 **IE3**

Задача

Вы хотите использовать файлы cookie, чтобы сохранять данные к следующему посещению страницы.

Решение

В секции «Обсуждение» приведена библиотека cookies.js, упрощающая запись и чтение данных cookie. Чтобы создать cookie, вызовите функцию setCookie(), передав ей в качестве аргументов имя cookie и строковое значение:

```
setCookie("userID", document.entryForm.username.value);
```

Чтобы узнать значение cookie, используйте функцию getCookie():

```
var user = getCookie("userID");
```

Обсуждение

Листинг 1.1 содержит код библиотеки cookies.js.

Листинг 1.1. Библиотека cookies.js

```
// Вспомогательная функция для формирования даты окончания действия
// в нужном формате. Три целочисленных параметра означают число дней.
// часов и минут, через которые истечет срок действия cookie. Все параметры
// обязательные, поэтому используйте нули там, где это нужно.
function getExpDate(days, hours, minutes) {
    var expDate = new Date();
    if (typeof days == "number" && typeof hours == "number" && typeof hours == "number") {
        expDate.setDate(expDate.getDate() + parseInt(days));
        expDate.setHours(expDate.getHours() + parseInt(hours));
        expDate.setMinutes(expDate.getMinutes() + parseInt(minutes));
        return expDate.toGMTString();
    }
}

// Вспомогательная функция, используемая функцией getCookie()
function getCookieVal(offset) {
    var endstr = document.cookie.indexOf (":", offset);
    if (endstr == -1) {
        endstr = document.cookie.length;
    }
    return unescape(document.cookie.substring(offset, endstr));
}

// Основная функция для определения значения cookie по имени
function getCookie(name) {
    var arg = name + "=";
    var alen = arg.length;
    var clen = document.cookie.length;
    var i = 0;
    while (i < clen) {
        var j = i + alen;
        if (document.cookie.substring(i, j) == arg) {
            return getCookieVal(j);
        }
        i = document.cookie.indexOf(" ", i) + 1;
        if (i == 0) break;
    }
    return null;
}
```

продолжение

Листинг 1.1 (продолжение)

```
// Сохраните значения cookie и некоторых дополнительных параметров
function setCookie(name, value, expires, path, domain, secure) {
    document.cookie = name + "=" + escape (value) +
        ((expires) ? "; expires=" + expires : "") +
        ((path) ? "; path=" + path : "") +
        ((domain) ? "; domain=" + domain : "") +
        ((secure) ? "; secure" : "");
}

// Уничтожение cookie путем установки ему прошедшего срока истечения
function deleteCookie(name,path,domain) {
    if (getCookie(name)) {
        document.cookie = name + "=" +
            ((path) ? "; path=" + path : "") +
            ((domain) ? "; domain=" + domain : "") +
            "; expires=Thu, 01 Jan 70 00:00:01 GMT";
    }
}
```

Листинг начинается со вспомогательной функции (getExpDate()), которая используется другими функциями для упрощения установки срока окончания действия. Вторая вспомогательная функция, getCookieVal(), применяется внутри библиотеки во время чтения cookie.

Функция getCookie() предназначена для чтения содержимого указанного cookie. Имя, передаваемое в качестве аргумента этой функции, представляет собой строку. Если браузер не может найти указанный cookie, функция возвращает пустую строку.

Для сохранения cookie служит функция setCookie(). Необходимые ей аргументы — имя cookie и его значение, которое будет сохранено. Если необходимо, чтобы cookie сохранился и после того, как пользователь закроет браузер, следует установить третий параметр — дату истечения действия. Чтобы обеспечить правильный формат даты, используйте функцию getExpDate().

Последняя функция, deleteCookie(), позволяет уничтожать cookie до того, как истечет срок их действия. Она просто устанавливает дату истечения на начало отсчета дат в JavaScript.

Библиотека может быть загружена в головном разделе документа:

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

При использовании библиотеки следует помнить, что все сохраняемые и восстанавливаемые значения представляют собой строки.

Использование файлов cookie — единственный способ сохранить данные на стороне клиента между посещениями странички. При этом сценарии могут читать только те записи, которые сделаны с их домена и сервера. Если в вашем домене несколько серверов, можно использовать пятый аргумент функции setCookie(), чтобы обеспечить доступ к cookie с разных серверов одного домена.

Обычно браузеры ограничивают число пар имя—значение двадцатью на один сервер, при этом одно значение не должно превышать 4000 символов, хотя на практике размер одной записи должен быть меньше 2000 символов. Другими словами, файлы cookie не являются средством хранения больших объемов данных

на стороне клиента. К тому же браузеры автоматически отсылают относящиеся к домену cookie при каждом обращении к странице. Поэтому желательно не злоупотреблять этой технологией, чтобы уменьшить нагрузку на пользователей, использующих модемы.

При сохранении пары имя—значение она остается в памяти браузера и сохраняется на диск при выходе из него, если дата истечения установлена где-то в будущем.

Поэтому не надо беспокоиться, если после записи cookie содержимое файла не изменилось. Разные браузеры по-разному хранят cookie (и в разных местах на каждой операционной системе). IE хранит cookie для каждого домена в отдельном текстовом файле, а Netscape помещает все в один текстовый файл.

Взаимодействие с cookie ведется через свойство document.cookie. Назначение библиотеки cookie.js — обеспечить более дружественный интерфейс для взаимодействия с этим свойством, которое не так удобно, как могло бы быть. Несмотря на то что при сохранении можно указывать несколько параметров, считать можно только один — значение cookie. Ни срок истечения действия, ни информация о домене недоступны.

Основное назначение cookie — сохранение пользовательских настроек между посещениями одной страницы. Сценарий в начале страницы читает cookie и, если они существуют, применяет прочитанные настройки к остальной части страницы. В рецепте 12.4 показано, как использовать эту методику для сохранения размера шрифта, выбранного пользователем. Например, такая команда сохраняет выбранный пользователем шрифт в cookie под названием fontSize, который будет существовать 180 дней после создания:

```
setCookie("fontSize", styleID, getExpDate(180, 0, 0));
```

Когда пользователь посетит страничку в следующий раз, это значение будет прочитано командой:

```
var styleCookie = getCookie("fontSize");
```

Прочитав это значение, сценарий должен применить к странице нужную таблицу стилей. Если же значение не было прочитано, следует использовать стиль, принятый по умолчанию.

Хотя в файлах cookie можно хранить только строчные значения, их можно использовать для хранения данных, обычно содержащихся в массивах или пользовательских объектах. В рецептах 3.12 и 8.14 показано, как преобразовать массив или произвольный объект в строчное значение, пригодное для хранения в cookie.

Смотрите также

Рецепт 10.4 показывает методику обмена данными между страницами с использованием cookie. В рецепте 12.4 показано, как использовать механизм cookie Для сохранения пользовательских настроек стиля страницы. В рецептах 3.12 и 8.14 показано, как преобразовать массив или произвольный объект в строчное значение, пригодное для хранения в cookie.

1.10. Преобразование символов строки в Unicode и обратно

NN2 IE3

Задача

Необходимо получить код символа в стандарте Unicode или наоборот.

Решение

Для получения кода символа по стандарту Unicode служит метод строчного значения charCodeAt(). Единственный параметр метода — целочисленный индекс символа в строке, для которого определяется код:

```
var code = myString.charCodeAt(3)
```

Если в строке содержится только один символ, для получения его кода необходимо передать методу 0:

```
var oneChar = myString.substring(12, 13);
var code = oneChar.charCodeAt(0);
```

Возвращаемое методом значение представляет собой целое число.

Чтобы произвести обратное преобразование, получив по номеру символа в кодировке Unicode сам символ, применяется метод fromCharCode(), принадлежащий статическому объекту String:

```
var char - String.fromCharCode 66);
```

В отличие от большинства других методов для работы со строками, этот метод должен вызываться только для объекта типа String, а не для строкового значения.

Обсуждение

Для латинских букв, цифр и знаков препинания коды ASCII совпадают с кодами Unicode. Но в отличие от ASCII Unicode содержит коды для многих национальных языков. Однако одного знания кодировки Unicode недостаточно, для отображения символов в операционную систему должна быть добавлена поддержка языка, к которому они принадлежат. Поэтому не стоит ожидать, что символ всегда будет отображаться в диалоговых окнах, полях ввода и на странице просто потому, что вы знаете его код. Если в системе не установлено обеспечение для отображения символов нужного языка, на экране вместо таких символов появятся вопросительные знаки или что-то подобное. Например, типичный компьютер в России не может отображать китайские иероглифы, пока на него не будет установлен набор шрифтов для китайского языка.

Смотрите также

В рецепте 1.2 показано, как извлекать подстроки длиной в один символ.

1.11. Кодирование и декодирование URL

NN6 **IE5.5(Win)**

Задача

Необходимо привести строку обычного текста к виду, пригодному для использования в URL, и наоборот, вернуть такой строке обычный вид.

Решение

Чтобы преобразовать строку, содержащую URL целиком, в совместимую с форматом URL форму, служит метод encodeURI(), получающий в качестве аргумента конвертируемую строку. Например:

```
document.myForm.action = encodeURI(myString);
```

Применяйте метод encodeURIComponent(), если вы формируете строку из фрагментов:

```
var srchString = "?name=" + encodeURIComponent(myString);
```

Для каждого из этих методов существует метод, производящий обратное преобразование:

```
decodeURI(encodedURIString);
decodeURIComponent(encodedURIComponentString);
```

Все эти методы не меняют содержимого исходной строки.

Обсуждение

С самых первых версий браузеров с поддержкой сценариев в них были доступны аналогичные по назначению функции escape() и unescape(), но современные стандарты не рекомендуют их использовать, отдавая предпочтение набору новых методов. Новые методы доступны в браузерах IE 5.5 и старше для Windows, и Netscape версии 6 и выше.

Методики кодирования, используемые новыми и старыми методами, слегка отличаются. Поэтому при декодировании всегда нужно использовать метод из того набора, который был использован при кодировании. Другими словами, если URL был кодирован с помощью метода encodeURI(), декодироваться он должен с помощью decodeURI().

Разница между методами encodeURI() и encodeURIComponent() определяется теми символами, которые они преобразуют в форму, пригодную для использования в URL. При таком преобразовании символ заменяется на знак процента (%), за которым следует шестнадцатеричный код символа в кодировке Unicode. Например, пробел преобразуется в %20. Обычные буквенно-цифровые символы не преобразуются, а специальные символы и знаки препинания кодируются по-разному разными методами. Метод encodeURI() преобразовывает следующие символы с ASCII-кодами от 32 до 126:

пробел " % < > [\ ] ^ - { | }

Например, при формировании простого поискового запроса сформированный URL необходимо пропустить через encodeURI(), чтобы удостовериться в том, что URL сформирован правильно:

```
var newURL = "http://www.megacorp.com? prod=Gizmo Deluxe"
location.href = encodeURI(newURL);
// После преобразования URL выглядит так:
//     http://www.megacorp.com?prod=Gizmo%20Deluxe
```

Метод encodeURIComponent() подвергает преобразованию гораздо больше символов, которые могут попасть в запрос из формы или из сценария. Символы, которые не кодируются методом encodeURI(), выделены жирным шрифтом в показанном ниже списке:

```
пробел " # $ % & + , / : ; < = > ? @ [ \ ] ^ ` { | }
```

Можно заметить, что перечисленные символы часто встречаются в сложных запросах, особенно ?, & и =. Поэтому такое преобразование допустимо применять только к значениям параметров запроса, а не к строке запроса целиком. Следует также отметить, что нельзя повторно пропускать подготовленный URL через encodeURI(), потому что символы % подвергнутся повторному преобразованию, что помешает серверу восстановить значения параметров.

Если, из соображений обратной совместимости, вам необходимо использовать метод escape(), необходимо иметь в виду, что он подвергает преобразованию еще большее количество символов:

```
пробел + \ " # $ % & ( ) * , / : ; < = > ? @ [ \ ] ^ ` { | } ~
```

Нужно также учитывать, что в Internet Explorer символ @ не кодируется методом escape().

Как видно из вышесказанного, очень важно использовать правильный метод декодирования, чтобы получить значения параметров в их исходном виде. Если строка, которую нужно декодировать, получена из внешнего источника (например, это часть строки поиска, возвращенной сервером), лучше всего разбить ее на пары имя–значение и декодировать только значение с помощью decodeURIComponent(). Такой способ позволяет получить наиболее корректный результат.

Смотрите также

Рецепт 10.6 показывает, как передавать данные от одной страницы к другой через URL, при этом используется кодирование строк.

1.12. Кодирование и декодирование строк в формате Base64

NN2 **IE3**

Задача

Необходимо преобразовать строку в кодировку Base64 и обратно.

Решение

В секции «Обсуждение» предлагается библиотека base64.js содержащая функции для таких преобразований. Чтобы кодировать строку в код Base64, используйте такую команду:

```
var encodedString = base64Encode("stringToEncode");
```

Для обратного преобразования:

```
var plainString = base64Decode("encodedString");
```

Обсуждение

Листинг 1.2 содержит код библиотеки base64.js.

Листинг 1.2. Библиотека base64.js

```
// Глобальные массивы с таблицами перекодировки Base64
var enc64List, dec64List;
// Заполнение массивов (однократное)
function initBase64() {
    enc64List = new Array();
    dec64List = new Array();
    var i;
    for (i = 0; i < 26; i++) {
        enc64List[enc64List.length] = String.fromCharCode(65 + i);
    }
    for (i = 0; i < 26; i++) {
        enc64List[enc64List.length] = String.fromCharCode(97 + i);
    }
    for (i = 0; i < 10; i++) {
        enc64List[enc64List.length] = String.fromCharCode(48 + i);
    }
    enc64List[enc64List.length] = "+";
    enc64List[enc64List.length] = "/";
    for (i = 0; i < 128; i++) {
        dec64List[dec64List.length] = -1;
    }
    for (i = 0; i < 64; i++) {
        dec64List[enc64List[i].charCodeAt(0)] = i;
    }
}
// Функция кодирования строки в кодировку Base64
function base64Encode(str) {
    var c, d, e, end = 0;
    var u, v, w, x;
    var ptr = -1;
    var input = str.split("");
    var output = "";
    while(end == 0) {
        c = (typeof input[++ptr] != "undefined") ? input[ptr].charCodeAt(0)
            ((end = 1) ? 0 : 0);
        d = (typeof input[++ptr] != "undefined") ? input[ptr].charCodeAt(0) :
            ((end += 1) ? 0 : 0);
```

продолжение

Листинг 1.2 (продолжение)

```
        e = (typeof input[++ptr] != "undefined") ? input[ptr].charCodeAt(0) .
            ((end += 1) ? 0 : 0);
        u = enc64List[c » 2];
        v = enc64List[(0x00000003 & c) « 4 | d » 4];
        w = enc64List[(0x0000000F & d) « 2 | e » 6];
        x = enc64List[e & 0x0000003F];

        // Выравнивание длины строки
        if (end >= 1) {x = "=";}
        if (end == 2) {w = "=";}

        if (end < 3) {output += u + v + w + x;}
    }
    // Форматирование строк длиной 76 символов по RFC
    var formattedOutput = "";
    var lineLength = 76;
    while (output.length > lineLength) {
       formattedOutput += output.substring(0, lineLength) + "\n";
       output = output.substring(lineLength);
    }
    formattedOutput += output;
    return formattedOutput;


function base64Decode(str) {
    var c=0, d=0, e=0, f=0, i=0, n=0;
    var input = str.split("");
    var output = "";
    var ptr = 0;
    do {
        f = input[ptr++].charCodeAt(0);
        i = dec64List[f];
        if ( f >= 0 && f < 128 && i != -1 ) {
            if ( n % 4 == 0 ) {
                c = i « 2;
            } else if ( n % 4 == 1 ) {
                c = c | ( i » 4 );
                d = ( i & 0x0000000F ) « 4;
            } else if ( n % 4 == 2 ) {
                d = d | ( i » 2 );
                e = ( i & 0x00000003 ) « 6;
            } else {
                e = e j i;
            }
            n++;
            if ( n % 4 = 0 ) {
                output += String.fromCharCode(c) +
                          String.fromCharCode(d) +
                          String.fromCharCode(e);
```

```
    while (typeof input[ptr] != "undefined");
    output += (n % 4 == 3) ? String.fromCharCode(c) + String.fromCharCode(d)
              ((n % 4 == 2) ? String.fromCharCode(c)  "");
    return output;
}

// Автоматическая инициализация глобальных переменных (при загрузке)
initBase64();
```

Сценарий начинается двумя объявлениями двух глобальных объектов, которые представляют собой таблицы перекодировки. Эти таблицы будут заполнены вызовом функции инициализации в конце сценария.

Используйте функцию **base64Encode()** для кодирования строки в кодировку Base64. Эта функция не меняет исходную строку. Для преобразования строки из кодировки Base64 в обычное представление предназначена функция base64Decode(), ее единственным аргументом является кодированная строка.

В Netscape 6 и более поздних версиях имеются методы для выполнения тех же действий. Метод atob() преобразует строку в кодировке Base64 в обычное представление, метод **btoa()** производит обратное преобразование. Но эти методы не являются частью стандарта ECMAScript, и неизвестно, когда они появятся в других браузерах и появятся ли вообще.

Откровенно говоря, большинство страниц, использующих сценарии, не нуждается в Base64 кодировании. Возможно причина этому — отсутствие готовых средств для работы с этой кодировкой. Кодировка Base64 использует очень маленький набор символов: a-z, A-Z, 0-9, +, / и =. Такая простая схема позволяет передавать преобразованные данные любого типа практически по любому сетевому протоколу. Приложения к электронным письмам могут кодироваться в Base64, чтобы их было возможно передавать через почтовые серверы. Почтовый клиент, получив такое письмо, декодирует текст и получает изображение, текст или исполняемый файл, прикрепленные к письму. Можно найти и другие применения этому методу кодирования. Дополнительную информацию по кодировке Base64 можно посмотреть по адресу: **http://www.ietf.org/rfc/rfc2045.txt.**

Смотрите также

Рецепт 1.11, демонстрирующий технику кодирования URL.

2 Числа и даты

2.0. Вступление

Разработчики дружественных к пользователю языков сценариев должны обязательно задумываться о тех, кто не является программистом. Но трудно себе представить, что такой язык будет принят профессиональными программистами, если в нем не будет серьезной основы. Математика может стать проклятием для тех, кто не учился программированию, но даже у такого общедоступного языка, как JavaScript, имеется солидный набор арифметических, тригонометрических и прочих операций, имеющихся в других языках. Поддерживаются в JavaScript и операции с датами, которые тоже весьма сложны. Данная глава включает в себя рецепты из обоих этих разделов.

Числа в JavaScript

Для большинства разработчиков сценариев особенности внутреннего представления чисел в JavaScript абсолютно неважны. Фактически, чем больше вы знаете о языках программирования и различных типах чисел, тем больше придется вам забыть в JavaScript. В отличие от большинства других языков, здесь есть только один тип числовых данных. Все целые и дробные значения представляются в JavaScript одним типом данных: number.

Внутри JavaScript все числа представлены в виде 64-битных чисел с плавающей запятой в формате IEEE. Такая точность обеспечивает представление чисел в диапазоне от 2.2E-208 до 21.79E+308. Эти граничные значения могут быть получены с помощью свойства статического объекта Number: Number.MIN_VALUE и Number.MAX_VALUE соответственно. Числа за максимальной границей рассматриваются в JavaScript как бесконечность. Бесконечность может также быть получена из свойств объекта Number: Number.NEGATIVE_INFINITY и Number.POSITIVE_INFINITY. Вряд ли эти свойства понадобятся в ваших сценариях, но язык поддерживает эти свойства ради полноты.

Числовые значения не содержат в себе никакой информации о форматировании. Если взять переменную, которая раньше содержала много чисел после запятой, то после арифметических преобразований, отбрасывающих дробную часть, незначащие нули отображаться не будут.

После строк (см. главу 1) числа являются самым распространенным типом значений в JavaScript. Так же как и строка, число может быть создано более формальным образом, с помощью конструктора Number. Таким образом, оба приведенных ниже выражения сформируют одинаковые значения:

```
var myNum = 55:
var myNum = new Number(55);
```

Но если проверить тип данных переменных (с помощью оператора **typeof),** окажется, что в первом случае переменная имеет тип number, а во втором — object. Числовое значение наследует свойства и методы объекта **Number,** большинство которых описано в данной главе.

Объект Math

Статический объект Math, доступный в любом контексте JavaScript, предоставляет набор стандартных методов и математических констант для работы с числами и тригонометрией. Ни один сценарий не должен создавать объект Math. Этот объект всегда существует и доступен для сценариев.

В табл. 2.1 перечислены свойства объекта **Math.** Все они представляют собой широко известные математические константы для использования в математических выражениях. Например, их можно применять, чтобы подсчитать длину окружности (диаметр, умноженный на ?), если диаметр записан в переменной d.

```
var circumference = d * Math.PI
```

Таблица 2.1. Свойства объекта Math

Свойство	Описание
E	Константа Эйлера (2,718281828459045)
LN2	Натуральный логарифм от 2 (0,931471805599453)
LN10	Натуральный логарифм от 10 (2,025850929940456840179914546843б)
L0G2E	Логарифм по основанию 2 от константы Эйлера
LOG10E	Логарифм по основанию 10 от константы Эйлера
PI	p (3,141592653589793)
SQRT1_2	Квадратный корень из 1/2
SQRT2	Квадратный корень из 2

Перечень методов объекта **Math** приведен в табл. 2.2. Часть из этих методов — тригонометрические функции, которые могут быть полезны, например, для анимации элементов. Другие методы предоставляют возможности, нужные время от времени, такие как возведение числа в степень, округление и получение максимума или минимума из пары двух чисел. Как и другие методы JavaScript, они не меняют значения переданных аргументов.

Таблица 2.2. Методы объекта Math

Метод	Описание
abs(x)	Возвращает абсолютную величину x
acos(x)	Возвращает арккосинус x (в радианах)
asin(x)	Возвращает арксинус x (в радианах)
atan(x)	Возвращает арктангенс x (в радианах)
atan2(x, y)	Возвращает арктангенс отношения y/x (в радианах)
ceil(x)	Возвращает ближайшее большее целое
cos(x)	Возвращает косинус x
exp(x)	Возвращает константу Эйлера, возведенную в степень x
floor(x)	Возвращает ближайшее меньшее целое
log(x)	Возвращает натуральный логарифм числа (по основанию e)
max(xl, x2)	Возвращает большее из двух чисел x1 и x2
min(xl, x2)	Возвращает меньшее из двух чисел xl и x2
pow(xl, x2)	Возвращает xl, возведенное в степень x2
random()	Возвращает псевдослучайное число в интервале от 0 до 1
round(x)	Если дробная часть числа больше или равна 0,5, то возвращает целую часть x, увеличенную на 1. В противном случае возвращает целую часть x
sin(x)	Возвращает синус x
sqrt(x)	Возвращает квадратный корень x
tan(x)	Возвращает тангенс x

Работа с датами и временем

С самого начала одним из самых мощных объектов в языке JavaScript был объект Date. Этот объект существует в каждом окне или фрейме и доступен в любой момент времени. Обычный способ работы с датами — через создание нового объекта Date с помощью конструктора:

```
var myDate = new Date():
```

Создание нового объекта Date (который дальше называется просто объектом даты) аналогично получению мгновенного снимка текущего времени. Объект даты содержит информацию не только о дате, но и о времени с точностью до миллисекунды, но этот объект — не часы, он не изменяет своего значения сам. Существует множество функций для получения отдельных компонентов даты и времени (год, месяц, день, число и т. д.). Аналогичный набор методов позволяет устанавливать значения отдельных компонентов даты и/или времени. Все эти методы представляют собой одну из возможностей производить вычисления с датами и временем.

Необходимо учитывать то, что объекты даты существуют на клиентском компьютере, который загрузил страницу. Не существует соединения с часами на сервере. Это означает, что все вычисления с датами зависят только от точности и настройки часов на стороне клиента. Влияние оказывает и временная зона, установленная на

компьютере, исполняющем сценарий. Неправильная настройка временной зоны, как показано в рецепте 15.8, может повлиять на вычисления с датой и временем.

ПРИМЕЧАНИЕ

При использовании Internet Explorer 5 для Macintosh возникают дополнительные трудности. Этот браузер считывает значение системных часов только при загрузке и не синхронизируется с ними в дальнейшем. Если компьютер переходит в спящий режим, внутренние часы IE останавливаются и находятся в таком состоянии до тех пор, пока компьютер не вернется в обычный режим. Будем надеяться, что в будущих браузерах не будет таких серьезных ошибок.

Если в сценарии нужны часы, отсчитывающие время, следует периодически создавать новый объект Date, чтобы получать текущее значение времени. В обсуждении рецепта 15.8 показано, как это сделать.

Некоторые особенности работы с датами в JavaScript могут вводить в заблуждение. Самое главное — понимать, что, когда вы создаете объект даты (представляющий текущий или какой-либо иной момент времени), данные хранятся в виде целого числа, представляющего собой число миллисекунд, прошедших с 1 января 1970 года. Основа для всех значений даты — всеобщее скоординированное время (UTC), что, по существу, то же самое, что и Гринвичское время (GMT). Основная сложность в том, что при просмотре значения времени, сохраненного в объекте, компьютер автоматически переводит время в локальное, несмотря на то что хранится Гринвичское время. Например, если создать объект даты на компьютере в Нью-Йорке в пятницу, 10:00 вечера, то в объекте будет записано Гринвичское время, соответствующее 3:00 утра в субботу. Но если потребовать у системы показать сохраненное значение, оно будет автоматически переведено в локальное время и показано как 10:00 вечера, пятница.

По большей части такое различие между внутренним содержимым и внешним представлением неважно. Поскольку все объекты даты на одном компьютере ведут себя одинаково, такие вычисления, как определение интервала времени, разделяющего две даты, дают одинаковые результаты. Задумываться о Гринвичском времени и часовых поясах необходимо только в том случае, если в вычислениях используется время в нескольких поясах. В рецепте 15.8 показано, как подсчитывать смещения между различными часовыми поясами.

Рецепты в этой главе содержат примеры основных вычислений с датой и временем. В главе 15 описываются дополнительные практические применения таких вычислений. Объект Date — мощный «зверь», приручив которого можно оживить возможности настройки и динамику страницы.

2.1. Преобразования между числами и строками

NN4 **IE4**

Задача

Необходимо преобразовать числовое значение в строку или наоборот.

Решение

Метод toString(), имеющийся у числового значения, преобразует его в строковое:

```
var numAsStringValue = numValue.toString();
```

Объект String с таким содержимым можно сформировать, передав число в качестве аргумента конструктору объекта:

```
var numAsStringValue = new String(numValue);
```

Для обратного преобразования строки в число используются методы parseInt(), если число целое, и parseFloat(), если оно может быть дробным:

```
var intValue - parseInt(numAsString. 10):
var floatValue = parseFloat(numAsString);
```

Оба этих метода работают во всех браузерах, поддерживающих сценарии.

Обсуждение

Во многих случаях интерпретатор JavaScript пытается автоматически преобразовывать числа в строки, и наоборот. Например, при умножении числа, представленного в виде строки, интерпретатор конвертирует строковое значение в числовое перед выполнением операции. Тем не менее такое преобразование не всегда происходит. Например, оператор сложения (+) играет в JavaScript двойную роль: сложение чисел и объединение строк. В том случае, когда один из операндов — строка, происходит конкатенация операндов как строк. Таким образом, в JavaScript выражение 2+"2" равняется "22".

Строку необходимо преобразовывать в число в том случае, когда необходимо выполнять математические операции над данными, введенными в поля формы. Свойство value любого поля ввода поддерживает только строковые значения. Например, чтобы сложить числа в двух полях и поместить результат в третье, необходимо до вычислений преобразовать оба значения в числа. Для того чтобы поместить число в поле ввода, достаточно просто присвоить его свойству value этого поля. JavaScript автоматически преобразует число в строку, потому что строка — единственный тип данных, подходящий для отображения в поле ввода. Пример кода:

```
var val1 = parseFloat(document.myForm.firstNum.value);
var val2 = parseFloat(document.myForm.secondNum.value);
var result = val1+ val2:
document.myForm.sum.value = result:
```

В отличие от большинства других языков программирования, JavaScript не различает числовые типы данных по свойствам. Число остается числом, будь оно целым или дробным. Единственное отличие функции parseInt() в том, что даже если в строке есть десятичная точка и разряды после нее, функция все равно возвратит целое значение. Такое поведение может быть удобным в том случае, когда необходимо проанализировать строку, которая начинается с числа,

но содержит дополнительные символы после него. Например, если свойство navigator.appVersion содержит следующий текст:

```
4.0 (compatible: MSIE 6.0: Windows 98: Q312461)
```

то, чтобы получить целое число в начале строки, можно воспользоваться командой:

```
var mainVer = parseInt(navigator.appVersion. 10):
```

Аналогично, если строка начинается с дробного числа, его можно получить, вызвав функцию parseFloat(). Другими словами, оба этих метода пытаются читать из строки символы до тех пор, пока могут интерпретировать их как число. Когда они встречают нечисловой символ, разбор строки прекращается и функция возвращает то, что она сумела прочитать.

Не забывайте устанавливать второй параметр parseInt() равным 10, показывая тем самым, что анализируется десятичное число. В противном случае, если строка начинается с 0 и с 8 или 9, считываемое число интерпретируется как восьмеричное (при этом цифры 8 и 9 являются нечисловыми). Метод parseFloat() всегда рассматривает строку как десятичное число (см. рецепт 2.6).

Что же касается преобразования числа в строку, есть старый трюк, используемый с самых ранних дней JavaScript. Это применение описанного выше поведения оператора сложения, который ожидает предпочтение конкатенации, если один из аргументов — строка. Если добавить пустую строку к числу, результат будет строковым представлением этого числа:

```
var numAsString = numVal + ""
```

Такая запись не слишком элегантна, зато компактна и обладает полной обратной совместимостью. Если вы увидите такой код, будете знать, что она означает.

Смотрите также

Рецепт 2.6 показывает способы преобразования чисел из одной системы счисления в другую.

2.2. Проверка правильности числа

NN3 IE4

Задача

Перед выполнением математических операций необходимо убедиться, что значение представляет собой число.

Решение

Лучший способ проверить значение, пришедшее из неизвестного источника, — использовать оператор typeof. Применив этот оператор к любому числовому зна-

чению, получим строку "number". Это можно использовать, например, следующим образом:

```
if (typeof someVal == "number") {
    //Обработка числового значения
}
```

Некоторые методы JavaScript, сигнализируя о невозможности выполнить вычисление, возвращают специальное значение, NaN («not a number» — не число). Если какой-либо из методов, ожидающих числовой аргумент, получает в качестве аргумента NaN, он, в свою очередь, тоже возвращает NaN. Например:

```
var myVal = parseInt(document.myForm.myAge.value);
if (isNaN(myVal)) {
    alert( Пожалуйста, проверьте поле \"Возраст\".");
} else {
    // Обработка числового значения
}
```

Обсуждение

Только что показанный пример может создать ложное впечатление о методе isNaN(). Этот метод не подходит для проверки корректности числа, введенного в текстовое поле на форме. Дело в том, что методы parseInt() и parseFloat() анализируют не всю строку, а только первые числовые символы. Например, если ввести в поле ввода возраста текст "32G", то метод parseInt() вернет значение 32, несмотря на то что число введено некорректно. Поэтому такое текстовое значение нельзя передавать, например, базе данных, ожидающей получить полностью корректное число. Смотрите описание более надежного метода проверки в рецепте 8.2.

Необязательно проверять тип всех переменных перед выполнением математических действий. Большинство значений в сценарии находятся под жестким контролем программиста, позволяя решать затруднения с типами данных до введения сценария в действие. Правда, необходимо с осторожностью относиться к случаям, когда пользователь вводит уравнения.

Значение NaN лучше всего рассматривать как вспомогательное средство при отладке. С его помощью можно определить ошибку в вычислениях и вывести диалоговое окно с информацией. В этом случае появление значения NaN в любой из переменных можно трактовать как ошибку.

В завершение нужно упомянуть, что значение NaN имеет числовой тип данных. Для того чтобы его получить, можно использовать одно из свойств статического объекта Number.

Смотрите также

В рецепте 8.2 показан способ проверки числовых данных, вводимых в форму.

2.3. Проверка равенства чисел

NN2 IE3

Задача

Прежде чем продолжать вычисления, необходимо выяснить, равны ли два числа.

Решение

Для проверки чисел на равенство служит стандартный оператор сравнения (==):

```
if (firstNum== secondNum) {
    // Числа равны
}
```

Оба сравниваемых значения могут быть как переменными, так и числовыми константами.

Обсуждение

В JavaScript есть два типа оператора сравнения. Стандартный, совместимый оператор **(==)** автоматически преобразует сравниваемые величины к одному типу. Рассмотрим переменные, заданные следующим образом:

```
var numA = 45:
var numb = new Number(45);
```

Эти переменные содержат одно и то же значение, но имеют разный тип данных. Первая переменная имеет числовой тип данных, в то время как вторая представляет собой объект Number. При сравнении этих двух переменных с помощью оператора == JavaScript производит различные преобразования типов, проверяя равенство значений. В данном случае равенство будет достигнуто, и оператор

```
numA = numb
```

вернет значение true.

Другой тип оператора сравнения, оператор строгого сравнения (===), не производит никаких преобразований типов. Если переменные определены так, как это показано выше, следующий оператор сравнения вернет false, потому что переменные имеют различный тип данных:

```
numA === numB
```

Если одно число — целое, а другое представляет собой целочисленное значение с плавающей точкой, как 4 и 4,0, оба оператора сравнения сочтут типы и значения переменных равными и вернут true. Все числа имеют одинаковый тип данных в JavaScript.

Для того чтобы определить неравенство двух величин, можно воспользоваться одним из двух операторов: проверки на неравенство **(!=)** или его строгой фор-

мой (!==). Например, если необходимо особо обработать некоторое значение, можно использовать следующее ветвление кода:

```
if(parseInt(document.myForm.myTextBox.value) != 0) {
    // Обработка ненулевого значения в поле ввода
}
```

Все, что сказано о преобразовании типов, в полной мере относится и к операторам проверки на неравенство.

Смотрите также

В рецепте 2.1 показана методика преобразования чисел в строки, и наоборот.

2.4. Округление чисел

NN2 IE3

Задача

Необходимо округлить значение с плавающей запятой до ближайшего целого числа.

Решение

Используйте метод Math.round():

```
var roundedVal = Math.round(floatingPointValue);
```

Эта операция не изменяет исходное значение, потому округленную величину нужно поместить в переменную.

Обсуждение

Алгоритм, используемый методом Math.round(), таков: если число меньше, чем x.5, то оно округляется до x, иначе — до x+1.

Объект Math имеет еще несколько полезных методов для преобразования дробных чисел в целые. Методы Math.floor() и Math.ceil() возвращают соответственно ближайшее меньшее и ближайшее большее целые числа. Например, Math.floor(3.25) вернет 3, а Math.ceil(3.25) вернет 4. То же верно и для отрицательных значений, хотя на первый взгляд кажется, что с ними должно быть все наоборот. Math.floor(-3.25) вернет -4, а Math.ceil(-3.25) вернет -3. Поэтому метод Math.floor() можно использовать для отбрасывания дробной части только для положительных значений.

В JavaScript переменная, которая содержала дробное число, может в следующем операторе стать целочисленной. Для программистов, привыкших к тому, что каждый тип чисел представляет собой отдельный тип данных, такое поведение может показаться раздражающим.

Смотрите также

Раздел «Объект Math» во вступлении к этой главе.

2.5. Форматирование чисел для вывода

NN2 IE3

Задача

Необходимо отобразить результат вычислений с фиксированным числом знаков после запятой.

Решение

Недавние добавления в язык JavaScript (и в стандарт ECMA) облегчают задачу вывода чисел с фиксированным числом десятичных знаков. Эти методы доступны в IE 5.5 для Windows и в Netscape 6 или позже. Метод toFixed() возвращает строку с фиксированным числом знаков справа после десятичной точки:

```
document.myForm.total.value = someNumber.toFixed(2):
```

Аргумент данного метода определяет число знаков, которые будут показаны после точки. Даже если число целое, этот метод отобразит его как дробное, с нулями после точки.

Чтобы ограничить общее число знаков строкового представления числа, применяется метод toPrecision():

```
document.myFom.rate.value = someNumber.toPrecision(5);
```

Аргумент этого метода определяет общее число знаков в записи числа, включая знаки как до точки, так и после нее. Если запись числа содержит меньше знаков чем требуется, число будет дополнено нулями справа:

```
var num = 98.6:

var preciseNum = num.toPrecision(5): // preciseNum станет равен 98.00
```

Форматирование чисел в старых браузерах сделано более неуклюже, поэтому лучше использовать вспомогательную функцию formatNumber(), показанную в секции «Обсуждение». При вызове функции необходимо передать ей число (или строку, которая будет преобразована в число) и указать количество знаков после запятой, которые будут использованы при отображении:

```
document.myForm.total.value = formatNumber(someNumber, 2):
```

Результат, возвращаемый этой функцией, предназначен для отображения на странице, а не для вычислений, поэтому если входные данные не подходят, сообщение об этом возвращается вместо числа.

Обсуждение

В листинге 2.1 показана функция formatNumber().

Листинг 2.1. Функция для отображения чисел formatNumer()

```
function formatNumber (num. decplaces) {
    // Преобразование в числовое значение. если получена строка
    num = parseFloat(num);
    // Удостоверяемся  что преобразование произошло нормально
    if (!isNaN(num)) {
        // умножаем число на 10 в степени decplaces
        // и округляем результат до ближайшего целого.
        // а затем преобразуем в строку
        var str = "" + Math.round (eval(num) * Math.pow(10.decplaces));
        // Если число получилось в экспоненциальной записи.
        // значит, оно слишком мало или велико
        if (str.indexOf("e") != -1) {
            return "Out of Range":
        }
        // дополняем число нулями слева. если это нужно
        while (str.length <= decplaces) {
            str = "0" + str:
        }
        // Вычисляем положение десятичной точки
        var decpoint = str.length - decplaces:
        // собираем результирующую с року из трех частей: (a) строка до точки:
        // (b) сама точка и (c) остаток строки
        // и возвращаем результат
        return str.substring(0.decpoint) + "." + str.substring(decpoint.str.length);
    } else {
        return "NaN":
    }
}
```

При использовании этих методов преобразования числа в строку необходимо понимать, как происходит округление чисел при отображении. Результат такого округления зависит от значения цифры, следующей после последней видимой. Например, если отобразить в строку число 1.2345 с двумя знаками после точки, получится 1.23, потому что цифра, следующая за 3, равна 4.

Функции, перечисленные в этом рецепте, просты и не делают многого из того, что может использоваться в программах. Они не умеют разделять разряды больших чисел или добавлять знаки валют в символ. Такие дополнительные возможности необходимо реализовывать самостоятельно.

Проще всего отделять разряды в записи большого числа с помощью регулярных выражений. Вот пример простой функции, вставляющей запятые в необходимых местах и не зависящей от размера числа (число должно быть целое, записанное обычным образом):

```
function formatCommas(numString) {
    var re = /(-?\d+)(\d{3})/:
    while (re.test(numString)) {
```

```
        numString = numString.replace(re, "$1.$2");
    }
    return numString;
}
```

Поскольку эта функция может обрабатывать только целые числа, проще всего использовать ее в программах, которые при выводе разделяют число на целую и дробную части. Вот как можно изменить последний оператор return в показанной выше функции formatNumber(), чтобы разряды в возвращаемом ей результате были бы разделены с помощью formatCommas():

```
return formatCommas(str.substring(0,decpoint)) + "." +
   str.substring(decpoint,str length);
```

Продолжая разговор о группировании разрядов, нужно упомянуть о том, что нередко пользователь может вводить числа именно в таком виде, удобном для чтения. Но числа в таком виде непригодны для передачи в базу данных или дальнейшей обработки. Чтобы решить эту проблему, можно использовать обработчик события onsubm.it, преобразовав число к обычному виду перед передачей. Функция для удаления запятых, использующая регулярные выражения, может выглядеть так:

```
function stripCommas(numString) {
    var re = /,/g;
    return numString.replace(re,"");
}
```

Существует еще один сравнительно новый метод объекта Number, предназначенный для форматирования числа. Этот метод называется toLocaleString() и вызывается так:

```
var formattedString = myNumber.toLocaleString();
```

Стандарт ECMAscript не рекомендует применять этот метод для получения какого-либо особого оформления, поскольку его результат в значительной степени зависит от того, как браузер пожелает выровнять число с учетом локализации. На сегодняшний момент только Internet Explorer производит какие-либо дополнительные действия при использовании этого метода. При этом все числа округляются до двух знаков после запятой. IE для Windows также разбивает разряды на группы запятыми, если число велико. Несмотря на то что Netscape 6 и более поздние версии поддерживают этот метод, они не производят никакого дополнительного оформления.

Нужно помнить о том, что в разных странах принято по-разному оформлять числа. Например, в Европе нередко используют для группировки разрядов точки, а дробную часть отделяют запятой. Если вы хотите использовать подобное форматирование, необходимо изменить показанные выше функции. Оператор замены в функции formatCommas() для этого нужно переписать так:

```
numString = numString.replace(re, "$1'$2");
```

А первый оператор stripCommas() изменить соответственно на:

```
var re = /'/g;
```

Возможно, вам покажется удобным изменить имена функции formatCommas() и stripCommas() на что-то соответствующее их содержанию. Все это — реализация того, что должен делать метод toLocaleString(). Теперь все зависит от разработчиков браузеров: договорятся ли они о переносимой реализации.

Смотрите также

В рецепте 8.3 показано, как использовать обработчик события onsubmit, чтобы произвести проверку правильности данных формы перед отправкой.

2.6. Преобразование между десятичной и шестнадцатеричной системами счисления

NN2 IE3

Задача

Необходимо преобразовать десятичное число в шестнадцатеричное или наоборот.

Решение

Несмотря на то что ядро JavaScript содержит методы для перехода от шестнадцатеричной записи к десятичному числу, обратное преобразование необходимо делать самостоятельно.

Для определения значения числа, записанного в шестнадцатеричной форме, применяется метод parseInt(), которому в качестве второго параметра передается 16:

```
var decimalVal = parseInt(myHexNumberValue, 16);
```

При этом строка myHexNumberValue может содержать как обычную шестнадцатеричную запись числа, так и запись в формате, используемом в JavaScript, при этом шестнадцатеричные цифры предваряются символами Ox или OX. Вот два примера, демонстрирующих эти формы записи:

```
var decimalVal = parseInt("1f", 16);
var decimalVal = parseInt( 0x1f"  16);
```

Для преобразования числа в его шестнадцатеричную запись можно использовать такую функцию:

```
function dec2Hex(dec) {
    dec = parseInt(dec, 10);
    if (!isNaN(dec)) {
        hexChars = "0123456789ABCDEF";
        if (dec > 255) {
            return "Out of Range";
        }
        var i = dec % 16;
        var j = (dec - i) / 16;
```

```
        result = "0x";
        result += hexChars.charAt(j) + hexChars.charAt(i);
        return result;
    } else {
        return NaN;
```

Поскольку JavaScript при выполнении вычислений преобразует все числа, независимо от формы записи, в свой внутренний формат, такое преобразование может быть нужно только при отображении шестнадцатеричного результата.

Обсуждение

Шестнадцатеричная арифметика мало используется в JavaScript, но язык обеспечивает минимальную поддержку чисел в таком формате. Шестнадцатеричное число можно обозначить с помощью символов 0x, после чего выполнять над ним любые математические операции. Следует только учитывать, что числа не содержат информации о той системе счисления, в которой они введены, поэтому результат вычислений с таким числом будет отображен как десятичное значение. Это позволяет использовать в одном выражении как десятичные, так и шестнадцатеричные числа:

```
var result = 0xff - 200;
```

Шестнадцатеричные цифры от a до f можно записывать в любом регистре.

Метод **parseInt()** удобен и для получения десятичного значения чисел в других системах счисления. Вот пример преобразования двоичного числа:

```
var decimalVar = parseInt("11010011", 2);
```

Смотрите также

Рецепт 2.1 показывает способы преобразования строк в числа и обратно.

2.7. Генерация псевдослучайных чисел

NN2 IE3

Задача

Необходимо получить псевдослучайное число.

Решение

Метод **Math.random()** возвращает псевдослучайное число между 0 и 1. Для получения целого псевдослучайного числа в заданном диапазоне (начинающемся в 0) можно использовать следующую формулу:

```
var result = Math.floor(Math.random() * (n + 1));
```

где n — это наибольшее из возможных случайных чисел. Чтобы получить числа в диапазоне, начинающемся с некоторого ненулевого значения, используйте следующую формулу:

```
var result = Math.floor(Math.random() * (n - m + D) + m:
```

где m — наименьшее, а n — наибольшее из генерируемых целых чисел.

Обсуждение

В показанных выше примерах внимание уделялось случайным целым числам, которыми можно имитировать, например, бросание кубика. Но ничто не мешает удалить вызов Math.round() и генерировать дробные случайные числа из заданного диапазона (следует также заменить n+1 на n).

Генератор случайных чисел в JavaScript не имеет механизма изменения значения счетчика случайных чисел для получения более достоверных результатов.

Смотрите также

Раздел «Объект Math» во вступлении к этой главе.

2.8. Вычисление тригонометрических функций

NN2 **IE3**

Задача

Необходимо использовать одну из тригонометрических функций, например, для анимации подвижного элемента.

Решение

Объект Math содержит набор основных тригонометрических функций. Каждая из них требует один аргумент, при этом все углы измеряются в радианах. Вот пример вычисления синуса:

```
var sineValue - math.sin(radiansInput);
```

Обсуждение

Краткое описание математических методов и констант смотрите во вступлении к этой главе. Применение тригонометрических функций для движения элемента по кругу можно увидеть в рецепте 13.10.

Смотрите также

Секцию «Объект Math» во вступлении к этой главе; рецепт 13.10, где показано, как применить тригонометрические функции для перемещения точки по кругу.

2.9. Использование объекта Date

NN2 IE3

Задача

Необходимо создать объект Date для вычислений или вывода на экран.

Решение

В конструкторе объекта Date можно указывать любые аргументы, обозначающие дату и время. Вот возможные варианты:

```
var myDate = new Date(yyyy, mm, dd, hh, mm, ss);
var myDate = new Date(yyyy, mm, dd);
var myDate = new Date("название месяца, день, год чч:мм:сс");
var myDate = new Date("название месяца, день, год");
var myDate = new Date(число миллисекунд с начала отсчета);
```

С помощью этих выражений можно создать объект, обозначающий практически любую дату (начиная примерно с 100 года нашей эры). Если при создании объекта не указывать время, то оно автоматически установится на начало дня.

Если же нужно создать объект, содержащий текущую дату и время, необходимо вызвать конструктор без аргументов:

```
var now = new DateO;
```

Точность, с которой объект Date определяет текущее значение времени, зависит только от точности часов на клиентском компьютере, настроек времени и, иногда, — от особенностей браузера. Определяющее значение имеет правильная установка часового пояса и перевода часов на летнее или зимнее время.

Обсуждение

Обратите внимание на то, что конструктор объекта Date не может воспринимать многие варианты сокращенной записи, такие как мм/дд/гг и многие другие. Вместо этого отдельные составные части даты и времени должны передаваться в конструктор раздельно. Например, ввод даты из формы может выглядеть так:

```
var dateEntry = new Date(document.myForm.year.value,
                         document.myForm.month.value,
                         document.myForm.date.value);
```

В этом примере JavaScript также автоматически преобразует строковые значения в числа, которые требуются конструктору.

Несмотря на то что ввод дат в формате мм/дд/гггг или мм-дд-гггг официально не должны поддерживаться, многие браузеры разрешают их использовать. Поэтому те, у кого возникла мысль использовать один из этих форматов, должны помнить, что никто не гарантирует работу такого кода во всех браузерах и операционных системах.

Важно понимать, что все манипуляции с объектом Date происходят целиком и полностью на стороне клиента. Никакого взаимодействия с часами или настройками часового пояса на сервере не происходит. В лучшем случае сервер может установить метку времени на странице в тот момент, когда она передается, но он ничего не сможет сделать с объектом Date на стороне клиента. Любые попытки синхронизировать объект Date с часами на сервере обречены из-за задержки между выдачей страницы сервером и ее отображением у клиента.

Реализация объекта Date в более ранних браузерах, чем IE 4 или Navigator 4, содержит множество ошибок, особенно при работе с часовыми поясами. Поэтому если вычисления с датами важны, нежелательно разрешать их выполнение на старых браузерах.

Смотрите также

В рецептах 2.10 и 2.11 показана методика вычислений с датами. В рецепте 2.12 показано, как использовать регулярные выражения для проверки правильности данных в форме. Рецепты 15.7 и 15.8 демонстрируют, как вычислить время, оставшееся до какого-то момента в будущем.

2.10. Вычисление прошедшей или будущей даты

NN2 **IE3**

Задача

Необходимо вычислить дату, основываясь на количестве дней до или после некоторой заданной даты.

Решение

Основной метод решения таков: необходимо создать объект даты с известным значением, а затем добавить или вычесть из него некоторое количество единиц времени. После этого можно получить числовое или строковое представление измененной даты, считывая ее отдельные компоненты.

Например, требуется вычислить дату через десять дней после текущей. Для этого создается объект, содержащий текущее значение времени, после чего к компоненту объекта, означающего день, прибавляется 10:

```
var myDate = new DateO:
myDate.setDate(myDate.getDate() + 10):
```

С этого момента объект myDate содержит информацию о будущей дате. Теперь можно получить строковое представление этой даты:

```
document.my Form.deadline value = myDate.toLocaleString();
```

Обсуждение

Увеличение и уменьшение даты работает даже тогда, когда это кажется нелогичным. Например, если объект даты указывает на двадцать пятое число месяца, то такая команда:

```
myDate.setDate(myDate.getDate()  +10):
```

передвинет дату на 10 дней вперед. Несмотря на то что 25 + 10 = 35, объект Date автоматически исправляет переданные ему значения и устанавливает дату в следующем месяце, через десять дней после текущей.

Используя на внутреннем уровне представление времени в виде числа миллисекунд, объект Date легко приспосабливается к границам месяцев и лет. Информация о числе, месяце и годе вычисляется только при запросе. Поэтому если прибавить десять дней к 25 июля (в этом месяце 30 дней), получится 5 июня, а если добавить те же десять дней к 25 июня (в нем 31 день), получим 4 августа. Обо всех сложностях с числом дней в месяце JavaScript заботится сам.

Объект Date содержит множество функций для работы с отдельными компонентами даты, начиная от года и кончая миллисекундами. В табл. 2.3 перечислены основные методы и границы их аргументов.

Таблица 2.3. Методы объекта Date

Метод чтения	Метод записи	Диапазон	Описание
getTime()	setTime(val)	0–...	Число миллисекунд с 1.01.1970, 00:00:00 по UTC
getSeconds()	setSeconds(val)	0-59	Число секунд
getMinutes()	setMinutes(val)	0-59	Число минут
getHours()	setHours(val)	0-23	Число часов
getDay()	setDay(val)	0–6	День недели (воскресенье = 0, понедельник = 1 и т. д.)
getDate()	setDate(val)	1-31	Число
getMonth()	setMonth(val)	0–11	Номер месяца
getFullYear()	setFullYear(val)	1970–...	Год (в полной записи)

Все эти методы работают с часовым поясом, установленным на локальном компьютере. Если необходимы более глобальные вычисления, использующие несколько часовых поясов, смотрите рецепт 15.8.

Смотрите также

Рецепт 2.9 — создание объекта Date. В рецепте 2.11 показано, как вычислить число дней между двумя датами. Рецепты 15.6-15.8 содержат дополнительные примеры применения вычислений с датами.

2.11. Определение числа дней межу двумя датами

NN2 IE3

Задача

Нужно определить, сколько дней разделяет две известные даты.

Решение

Для такого вычисления можно применить функцию daysBetween(), текст которой приведен в секции обсуждения. Вот пример ее использования:

```
var projectLength = 0;
// Проверяем данные в форме с помощью функции checkDate() из рецепта 2 12
var startField « document.entryForm.startDate;
var endField - document.entryForm.endDate;
if (checkDate(startField) && checkDate(endFie d)) {
    var startDate - new Date(startField.value);
    var endDate - new Date(endField.value);
    projectLength - daysBetween(startDate, endDate);
}
if (projectLength > 0) {
    alert("Вы указали " + projectLength + " дней на проект ");
}
```

Обсуждение

Листинг 2.2 содержит текст функции daysBetween(). Оба аргумента этой функции должны быть объектами Date.

Листинг 2.2. Функция daysBetween() для вычисления числа дней между двумя датами

```
function daysBetween(date1, date2) {
    var DSTAdjust - 0;
    // Это константа, используемая в вычислениях ниже
    oneMinute = 1000 * 60;
    var oneDay - oneMinute * 60 * 24;
    // Если вместе с датой указано время, сделаем его равным
    date1.setHours(0);
    date1.setMinutes(0);
    date1.setSeconds(0);
    date2.setHours(0);
```

```
    date2.setMinutes(0);
    date2.setSeconds(0);
    // учтем задержку, вносимую переводом часов на летнее и зимнее время
    if (date2 > date1) {
        DSTAdjust =
            (date2.getTimezoneOffset()  date1.getTimezoneOffset()) * oneMinute;
    } else {
        DSTAdjust =
            (date1.getTimezoneOffset()  date2.getTimezoneOffset()) * oneMinute;
    }
    var diff = Math.abs(date2.getTime() - date1.getTime()) - DSTAdjust;
    return Math.ceil(diff/oneDay);
}
```

Вычисление основывается на определении числа миллисекунд, разделяющих даты. Для того чтобы точно вычислить число дней между двумя датами, время, если оно указано, игнорируется путем установки в 0.

Возможно, вы обратили внимание на тот участок кода, который вычисляет значение переменной DSTAdjust. Эта переменная служит для учета переходов на летнее и зимнее время, используемых во многих странах. Несмотря на то что каждый день содержит фиксированное число миллисекунд, за счет перевода часов могут появиться дни, содержащие 23 или 25 часов. Потому разница между двумя датами может оказаться больше или меньше на один час. Без учета этой разницы количество целых дней может быть определено неправильно.

Кажется почти волшебным то, что интерпретатор JavaScript, работающий в контексте определенной операционной системы, знает о том, что в течение одного времени года для данной страны нужно использовать одно смещение от гринвичского времени, а в другое время — другое. Именно это свойство используется предложенной функцией daysBetween() для определения поправки. Для интервалов времени, не включающих в себя перевод часов, поправка равна нулю. В противном случае поправка содержит число минут, на которые были переведены часы за этот интервал (потому что метод getTimezoneOffsetO выдает значение в минутах).

Смотрите также

Рецепт 15.7 содержит пример динамического обновления числа дней до Рождества. Рецепт 15.8 показывает, как сделать таймер, отсчитывающий время.

2.12. Проверка правильности даты

NN4 IE4

Задача

Необходимо проверить правильность даты, которую пользователь ввел в поле формы.

Решение

Для решения этой задачи можно применить функцию checkDate(), предложенную в секции обсуждения. Эта функция воспринимает дату, записанную в одной строке в формате мм/дд/гггг или мм-дд-гггг. Вот пример использования этой функции в обработчике события onchange для проверки введенной пользователем даты:

```
function validateDate(fld) {
    if (!checkDate(fld)) {
        // переход на данное поле ввода, если дата введена с ошибкой
        fld.focus0:
        fld.select():
    }
```

Обсуждение

Прежде чем проверять правильность даты, необходимо четко себе представлять, что должен ввести пользователь, назначение этой величины, что необходимо сообщить пользователю при ошибке. Достаточно просто проверить, записаны ли цифры на нужных местах. Однако такой метод проверки недостаточно хорош. В конце концов, кто-то может попытаться ввести, например, 45 января.

Функция проверки checkDate() на листинге 2.3 предполагает, что пользователь вводит дату в формате мм/дд/гггг или мм-дд-гггг. Функция проверяет, действительно ли дата введена корректно. Она не проверяет диапазон вводимых величин, потому можно ввести практически любой год. Поскольку эта функция проверяет данные, введенные в форму, она получает в качестве единственного аргумента ссылку на поле ввода (как и другие функции проверки, приведенные в рецепте 8.2). Если значение поля успешно прошло проверку, функция возвращает true, в противном случае пользователь получает сообщение с информацией об ошибке.

Листинг 2.3. Простейшая функция проверки даты

```
function checkDate(fld) {
    var mo. day. yr:
    var entry - fld.value;
    var re - /\b\d{1.2}[\/-]\d{1.2}[\/-]\d{4}\b/:
    if (re.test(entry)) {
        var delimChar - (entry.indexOf("/") !- -1) ? "/" : "-":
        var deli ml - entry.indexOf(delimChar);
        var delim2 - entry.lastIndexOf(delimChar);
        mo - parseInt(entry.substring(0, delim1), 10):
        day - parseInt(entry.substring(delim1+1, delim2). 10):
        yr - parseInt(entry.substring(delim2+1), 10):
        var testDate = new Date(yr, mo-1, day):
        alert(testDate)
        if (testDate.getDate() == day) {
            if (testDate.getMonth() + 1 = mo) {
                if (testDate.getFullYear() ==yr) {
                    return true:
                } else {
                    alert("Дата введена с ошибкой."):
```

```
            } else {
                alert("Месяц записан с ошибкой "):
            }
        } else {
            alert("Число указано с ошибкой."):
        }
    } else {
        alert("Неправильный формат даты  Используйте формат мм/дд/гггг.");
    }
    return false:
}
```

Первое действие, выполняемое функцией, это проверка формата введенной даты с помощью регулярного выражения. Если формат даты правилен, функция создает из отдельных компонентов объект Date, после чего отдельные компоненты этого объекта сравниваются со введенными. Если совпадения нет, делается вывод о неправильно введенных данных. Эта методика использует тот факт, что JavaScript воспринимает значения дат, выходящие за допустимые границы, и преобразует такие значения в корректные. Например, если пользователь вводит 2/30/2003, то интерпретатор исправляет эту некорректную дату и формирует дату 3/2/2003. Поскольку во введенной и в получившейся датах не совпадают значения числа и месяца, можно сделать вывод о том, что пользователь ввел некорректное значение.

Несмотря на то что функция использует регулярные выражения для проверки формата значения, она применяет также и более примитивные методы разбора строки для более детального анализа. Такая тактика необходима для преодоления неполной реализации расширенных регулярных выражений в некоторых браузерах, предшествующих IE 5.5 для Windows. Благодаря этому функция checkDate() корректно работает на всех основных браузерах с версии 4.

В том случае, когда необходимо вводить большие количества данных и продуктивность работы пользователя измеряется в количестве нажатий на клавиши, желательно сделать форму более интеллектуальной. Например, можно обеспечить возможность вводить год двумя цифрами, при этом процедура проверки должна восстанавливать год целиком, что требуется для ввода в базу данных. Более того, механизм сокращенного ввода не должен требовать периодической перенастройки по мере хода лет. Листинг 2.4 демонстрирует улучшенную версию функции checkDate(), содержащую эти улучшения. Изменения в коде выделены жирным шрифтом.

Листинг 2.4. Улучшенная функция проверки даты

```
function checkDate(fld) {
    var mo. day. yr:
    var entry = fld.value;
    var reLong = /\b\d{1,2}[\/-]\d{1,2}[\/-]\d{4}\b/:
    var reShort = /\b\d{1,2}[\/-]\d{1,2}[\/-]\d{2}\b/:
    var valid = (reLong.test(entry)) || (reShort.test(entry)):
    if (valid) {
        var delimChar = (entry.indexOf("/") != -1) ? "/" : "-":
        var delim1 - entry.indexOf(delimChar);
        var delim2 = entry.lastIndexOf(delimChar);
        mo - parseInt(entry.substring(0. delim1). 10):
```

продолжение

Листинг 2.4 (продолжение)

```
        day = parseInt(entry.substring(delim1+1, delim2), 10);
        yr - parseInt(entry.substring(delim2+1), 10);
        // Обработка сокращенной записи года
        if (yr < 100) {
            var today = new Date();
            // определяем нижнюю границу текущего века (например, 2000)
            var currCent = parseInt(today.getFullYear() / 100) * 100;
            // годы до текущего + 15 считаются принадлежащими текущему веку
            var threshold = (today.getFullYear() + 15) - currCent;
            if (yr > threshold) {
                yr += currCent - 100;
            } else {
                yr += currCent;
            }
        }
        var testDate - new Date(yr, mo-1, day);
        if (testDate.getDate() — day) {
            if (testDate.getMonth() + 1 == mo) {
                if (testDate.getFullYear() == yr) {
                    // Помещаем в поле дату в формате,
                    // подходящем для базы данных
                    fld.value - mo + "/" + day + "/" + yr;
                    return true;
                } else {
                    alert("Дата введена с ошибкой.");
                }
            } else {
                alert("Месяц записан с ошибкой.");
            }
        } else {
            alert("Число записано с ошибкой.");
        }
    } else {
        alert("Неправильный формат даты. Используйте формат мм/дд/гггг.");
    }
    return false;
}
```

Можно обойти множество потенциальных проблем с вводом даты, включая те из них, которые связаны с различиями в традициях, используя три отдельных поля ввода (или три списка) для ввода месяца, дня и года. Между тем, использование списков вместо полей ввода не освобождает от необходимости проверки данных в форме, потому что пользователь может ввести неправильную комбинацию из допустимых значений (например, что-нибудь наподобие 31 июня). Можно решить эту проблему, используя динамические формы и заполняя список дат только допустимыми для указанного месяца значениями (см. рецепт 8.13).

Поля для ввода даты составляют важную часть формы, в которой они расположены. Не стоит экономить на них.

Смотрите также

Рецепт 8.2 содержит дополнительные функции для проверки данных в форме.

3 Массивы и объекты

3.0. Вступление

Большинство задач, встречающихся в программировании, включают в себя перемещение данных в памяти. Большая часть данных, используемых в работе JavaScript, состоит из объектов, являющихся частью документа. Нередко сценарии, исполняемые на клиентском компьютере, сопровождаются данными, передаваемыми сервером или вмонтированными в сценарий (в виде массивов или пользовательских объектов). Возможно, вам покажется удобным использовать более гибкие структуры данных, отображающие содержимое страницы. Например, гораздо проще сначала сортировать данные таблицы в массиве, после чего вносить их в таблицу.

Одной из самых сложных задач, решаемых программистом, является проектирование структур данных, с которыми работают сценарии. Не принято планировать работу сценария без рассмотрения основных структур данных, с которыми он будет работать. Массивы и пользовательские объекты в JavaScript являются контейнерами и служат для организации хранения данных. Эти контейнеры обеспечивают простой способ хранения данных и их структурирования.

Массивы в JavaScript

Нестрогая типизация, распространенная в JavaScript, оказала свое влияние и на массивы. В отличие от аналогичных структур в других языках программирования, массивы в JavaScript не ограничены неким фиксированным размером, установленным при создании. Можно добавлять или удалять элементы из массива по своему желанию, это исключительно гибкие хранилища данных.

Другой особенностью массивов в JavaScript является то, что любой элемент массива может содержать данные любого типа. Можно смешивать в одном массиве строки, числа, булевские значения или объекты. Можно в любой момент менять значение любого элемента и даже его тип данных. Это не особенно разумно с точки зрения стиля программирования, но тем не менее возможно.

Индексы массивов являются целыми числами с отсчетом от нуля. Другими словами, для доступа к первому элементу массива myArray необходимо использовать запись myArray[0]. Для того чтобы изменить значение элемента массива, необходимо использовать простой оператор присваивания (=). Можно также

применять оператор присваивания со сложением, чтобы увеличить число или присоединить к строке текст.

Простейшие массивы в JavaScript одномерные, но, как будет показано в рецептах 3.2 и 3.9, можно создавать и более сложные структуры, включая многомерные массивы (массивы из массивов) и массивы из сложных пользовательских объектов.

Пользовательские объекты в JavaScript

Нестрогость JavaScript, проявляющаяся в работе языка с типами данных, массивами и значениями переменных, проявляется и в концепции объектов. Забудьте о том, что вы знаете о технологии объектно-ориентированного программирования и отношениях между объектами. Традиционные понятия классов, подклассов и передачи сообщений имеют мало отношения к JavaScript (хотя эти концепции могут появиться в языке в будущем). Вместо этого нужно рассматривать объект как контейнер для данных, в котором каждая запись имеет название для более удобного использования. Синтаксис использования пользовательских объектов в JavaScript подобен синтаксису, используемому при работе с другими объектами в JavaScript и DOM. Здесь используется то же «правило точки» (то есть myObject.myProperty и myObject.myMethod()).

Объектно-ориентированный подход опасен тем, что у вас может возникнуть желание превратить каждый фрагмент данных в отдельный объект. Это может привести к тому, что усилия, потраченные на формирование объекта, сведут на нет все преимущества функциональности и организации кода, которые обеспечивает данный подход. При объектно-ориентированном подходе простая задача нередко запутывается большим количеством связей между данными. Но если сценарий часто обращается к одним и тем же глобальным переменным, имеющим отношение к одной сущности, есть смысл организовать эти данные в объект. В последующих главах книги вы увидите множество объектов, служащих хранилищами данных, относящихся к отдельному элементу. Например, это могут быть элементы меню.

Несмотря на высказанные выше предупреждения, касающиеся различий между объектами в JavaScript (базирующихся на концепции наследования прототипа) и объектами в настоящих объектно-ориентированных средах, можно имитировать большую часть основных идей ООП. Рецепт 3.12 демонстрирует несколько таких примеров.

Массивы или объекты?

Итак, в каком же случае следует использовать массив, а в каком — объект? Массив следует рассматривать как нумерованный список маленьких элементов данных. Список сам по себе определяет назначение хранящихся в нем данных, например, это может быть список имен сотрудников или названий книг на полке. Положение одного элемента относительно другого при этом непринципиально, например, может возникнуть необходимость вывести список по алфавиту или упорядочить данные каким-то другим способом.

Хранение данных в таком «слепом» списке означает что когда-то потребуется перебирать все его элементы, возможно, выводя их значение в элемент HTML.

С другой стороны, объект больше подходит для объединения данных об одной сущности. Объект «сотрудник» может содержать информацию об имени и возрасте человека, объект «книга» может иметь самые разные свойства, такие как автор, издатель, категория книги и тому подобное. Каждое свойство имеет собственное имя для того, чтобы можно было получить прямой доступ к значению свойства (например, book.publisher). Можно снабдить объект набором методов, которые будут выполнять основные действия с ним (см. рецепт 3.8).

Как можно видеть на примере рецепта 3.7, иногда есть смысл применить массив из объектов. Использование массива обеспечивает сценарию возможность перебирать список объект за объектом, а применение объектов позволяет тому же сценарию просматривать отдельные свойства элементов списка. Например, имея список объектов «книга», можно просматривать весь список, составляя перечень книг, чьи авторы удовлетворяют заданному критерию.

Будьте готовы использовать массивы и объекты в любых комбинациях. Не только массив может состоять из объектов, но и отдельное свойство объекта может быть массивом. Например, объект, представляющий книгу, может хранить авторов в массиве, обеспечивая возможность хранить несколько авторов для каждой книги. Те книги, у которых только один автор, будут содержать в этом массиве только один элемент.

Насколько сухим кажется на первый взгляд название этой главы, настолько же важной является она для всей книги. Большинство рецептов этой главы являются базовыми строительными блоками для DHTML и сценариев и неоднократно используются в последующих частях книги.

Передача данных странице

Большая часть рецептов этой главы демонстрируют работу с массивами и объектами, встроенными непосредственно в код сценария на странице. Такой подход подходит для фиксированных наборов данных или данных, вводимых пользователем через форму. Но с помощью серверных программ можно применять и обновляемые источники, такие как внешние базы данных.

При использовании средств серверного программирования (таких как ASP, JSP, ColdFusion, PHP и многих других), формирующих страницы HTML с помощью шаблонов и серверных программ, можно преобразовывать наборы данных полученные по запросу из базы данных в массивы и объекты JavaScript, внедренные в страницу. Другой подход состоит в возможности загрузки внешних файлов сценариев (с помощью атрибута src тега <script>), расположенных на сервере. Передаваемый серверу URL сценария должен в этом случае содержать параметры, используемые для выборки данных из базы. Получив такой запрос, сервер производит выборку из базы данных и преобразует ее результат в объекты и массивы JavaScript, передаваемые в файле .js. Полученные данные становятся такой же частью страницы, как если бы они были непосредственно встроены в нее.

Возможны и более проработанные схемы, использующие, например, невидимый фрейм, непрерывно запрашивающий часто изменяющиеся данные на сервере и обновляющий связанные с ними страницы Невидимый Java-апплет может устанавливать соединение с серверным процессом и взаимодействовать

с данными на странице (это возможно в браузерах, поддерживающих технологию LiveConnect — связь Java и JavaScript). Все эти подходы преследуют одну цель: внедрить данные в массивы JavaScript и/или объекты, чтобы обеспечить клиенту возможность доступа к ним.

3.1. Создание простого массива

NN3 IE4

Задача

Необходимо создать обыкновенный массив данных.

Решение

В JavaScript есть два пути создания и заполнения массива: полный и упрощенный. Если не указывать никаких параметров конструктору, он создает пустой массив, который затем можно заполнять данными, элемент за элементом:

```
var myArray = new Array();
myArray[0] = "Alice"
myArray[1] = "Fred";
```

Размер массива указывать необязательно, но это можно сделать, передав конструктору единственный целочисленный параметр:

```
var myArray = new Array(12);
```

Этот оператор создает массив с 12 элементами, которые изначально заполняются значением null.

Если передать конструктору массива больше одного аргумента, разделив их запятой, он воспримет их как данные, которыми нужно заполнить массив. Так, следующий оператор создаст массив из трех элементов, каждый из которых представляет собой строку:

```
var myArray = new Array("Alice". "Fred". "Jean");
```

Сокращенный способ создания массива предполагает использование квадратных скобок вместо вызова конструктора:

```
var myArray = ["Alice". "Fred". "Jean"];
```

Сокращенный синтаксис можно использовать в IE 4, NN 4 и выше.

Обсуждение

После того как массив создан (с применением любого из вариантов записи), новые записи в него можно добавлять, присваивая значения следующим незаполненным элементам. Если размер массива, в который нужно добавить элемент, неизвестен, можно воспользоваться свойством массива length для определения

положения добавляемого элемента. Так как значение свойства length всегда на единицу больше, чем индекс последнего элемента, его можно напрямую использовать как индекс при добавлении новых записей:

```
myArray[myArray.length] = "Steve":
```

Такую конструкцию можно брать для заполнения данными любого массива, в том числе и пустого. Это особенно удобно, если вы время от времени изменяете код Вместо того чтобы жонглировать фиксированными номерами в бесчисленном количестве операторов присваивания, гораздо проще использовать свойство length и позволить элементам самим заполнять массив в нужном порядке. Индексы сами позаботятся о себе, даже если вы измените порядок заполняющих массив операторов.

Смотрите также

Рецепт 3.2 демонстрирует создание более сложного массива. В рецепте 3.8 имеется обсуждение использования массивов из объектов. Рецепт 3.8 поможет преобразовать массив в строковое значение.

3.2. Формирование многомерного массива

NN3 IE4

Задача

Нужно объединить данные в массив, имеющий два измерения (такой как таблица) или большее число измерений

Решение

Сформируем массив из массивов. В качестве примера рассмотрим небольшую таблицу с данными о продажах в регионах.

Описание	1	2	3	4
Восточный	2300	3105	2909	4800
Центральный	1800	1940	2470	4350
Западный	900	1200	1923	3810

Для того чтобы поместить эти данные в массив из трех элементов (представляющих отдельные строки таблицы), каждый из которых, в свою очередь, является массивом из четырех вложенных значений, можно использовать множество различных способов записи. В самом длинном варианте сначала отдельно создаются три массива, представляющих собой строки, после чего они объединяются в один общий массив:

```
var eastArray = new array(2300. 3105. 2909. 4800):
var centralArray = new Array(1800. 1940. 2470. 4350):
```

```
var westArrah = new Array(900. 1200. 1923. 3810):
var salesArray = new Array(eastArray, centralArray. westArray):
```

Самая компактная форма записи предусматривает исключительно сокращенную запись конструктора:

```
var salesArray = [[2300. 3105. 2909. 4800].
                  [1800. 1940. 2470. 4350].
                  [900. 1200. 1923. 3810]]:
```

Для доступа к элементам подобных двумерных массивов необходимо применить два индекса. Например, чтобы получить величину продаж в восточном регионе за первый квартал, можно использовать следующую запись:

```
salesArray[0][0]:
```

Между квадратными скобками в такой записи не должно быть никаких запятых или других знаков. Первый индекс относится к массиву верхнего уровня, а второй — к вложенному. Таким образом, чтобы выяснить продажи в центральном регионе за третий квартал, можно написать такой код:

```
salesArray[l][2]:
```

Данные в многомерных массивах можно читать и писать точно так же, как и в обычных.

Обсуждение

Число уровней вложения при создании многомерных массивов практически не ограничено. Каждое новое измерение увеличивает количество индексов и скобок, нужных для доступа к элементу. В рецепте 3.4 показано, как можно применить операторы цикла для перебора всех элементов сложного массива большой степени вложенности.

Одна из возможных проблем при использовании многомерных массивов — опасность забыть, какой смысл вкладывается в каждое измерение. В показанных выше примерах индексы отделены от их значения, региона или квартала. Все, что означают эти числа, — положение элемента в двумерном массиве. Программист сам обязан следить за тем, какой смысл вкладывается в каждое из чисел. Поэтому во многих случаях лучшим выходом было бы использование массива объектов. Свойства объектов имеют собственные имена, дающие контекст использования их данных. Рецепты 3.8 и 3.9 содержат дополнительные идеи на эту тему.

Смотрите также

Перебор всех элементов одно- и многомерного массива демонстрируется в рецепте 3.4. Рецепт 3.8 показывает, как использовать массивы объектов вместо многомерных массивов. Рецепт 3.9 предлагает реализацию пользовательского объекта для показанного выше примера с продажами, а также описывает, как использовать хэш-таблицы для ускорения доступа к отдельным элементам.

3.3. Конвертирование массивов в строки

NN3 IE4

Задача

Необходимо получить данные массива в виде строки или преобразовать данные из строки в массив.

Решение

Массивы и строки в JavaScript имеют набор методов, упрощающих преобразования между двумя этими типами данных, тем самым позволяя передавать массивы в другие страницы, используя механизм cookie или параметры в URL.

Прежде чем преобразовывать массив (одномерный) в строку, необходимо определить символ, который будет служить разделителем записей в строке. Этот символ не должен появляться нигде в данных массива. Чтобы преобразовать массив в строку, необходимо указать этот символ как единственный параметр метода массива join(). Следующий оператор преобразует массив в строку, взяв в качестве разделителя запятую:

```
var arrayAsString = myArray.join(",");
```

При выполнении этой операции исходный массив не меняется.

Если имеется строка, содержащая разделенные заданным символом данные, которую необходимо преобразовать в массив, необходимо передать символ-разделитель в качестве аргумента метода split() для строкового значения:

```
var restoredArray = myRegularString.split(" ");
```

Метод split() выполняет работу конструктора массива, автоматически заполняя его данными из строки.

Обсуждение

Несмотря на то что в вышеприведенных примерах в качестве разделителя используется один символ, можно взять строку, состоящую из любого количества символов. Например, если необходимо превратить данные массива в вертикальный список, помещенный в элемент textarea, в качестве разделителя можно применить специальный символ \n, который вставит разрыв строки между элементами массива. Аналогично, если предполагается поместить данные массива в виде списка в документ XHTML, в качестве разделителя можно использовать строку
. Для того чтобы отобразить такой список на странице, полученное строковое значение нужно присвоить свойству innerHTML элемента страницы.

Метод join() можно выбирать для одномерных массивов. Для многомерных массивов метод можно применять только к наиболее глубоко вложенным массивам, которые, в свою очередь, являются одномерными.

Еще больше возможностей предоставляет метод **splitQ.** В качестве разделителя для него можно использовать регулярное выражение. В качестве примера

рассмотрим строку, содержащую последовательность денежных сумм, разделенных запятыми:

```
var amounts = "30.25.120.00.45.09.200.10";
```

Если необходимо создать массив, состоящий только из целых частей этих сумм, можно рассматривать в качестве разделителя регулярное выражение, совпадающее с точкой, за которой следуют две цифры и необязательная запятая (потому что в последней записи запятой нет):

```
var amtArray = amounts.split(/\.\d{2},?/);
```

В том случае, когда разделитель встречается в конце строки, метод split() создает в конце массива элемент, соответствующий несуществующей записи, которая должна идти за разделителем. Обычно разделитель после последнего элемента не ставится, но если он присутствует, следует учитывать этот пустой элемент массива.

Метод split() имеет второй необязательный аргумент, обозначающий количество записей, которые должны быть помещены в массив из строки. Используя этот параметр, можно, точно указав количество элементов, содержащихся в строке, избежать проблемы с незаполненным элементом после завершающего разделителя. Этот параметр не является частью стандарта ECMAScript, но поддерживается всеми основными браузерами.

На практике преобразование массивов в строки ограничено массивами, содержащими только простые данные, такие как строки, числа и булевские значения. Элементы массива, содержащие ссылки на объекты (неважно, пользовательские это объекты или DOM), обычно не имеют осмысленного строкового представления. Для массива объектов DOM можно поместить в строку идентификаторы этих объектов. Если элементы страницы не изменятся, идентификаторы могут быть использованы для восстановления ссылок на элементы. Рецепты 3.13 и 8.14 содержат советы по преобразованию объектов в строки.

Смотрите также

Рецепт 3.13 содержит способ преобразования массива, содержащего пользовательские объекты и другие массивы, в строку и последующего восстановления исходного массива из строки. Рецепт 8.14 показывает, как преобразовать данные из формы в единую строку для передачи в другую страницу.

3.4. Работа с элементами массива

NN2 **IE3**

Задача

Необходимо перебрать все элементы массива и определить их значения.

Решение

Для перебора элементов массива можно использовать цикл for с увеличивающимся счетчиком, ограниченный размером массива. Следующий код, несмотря

на некоторую непрактичность, демонстрирует поочередное обращение ко всем элементам массива в цикле:

```
var myArray = ["Alice", "Fred", "Jean", "Steve"];
for (var i = 0; i < myArray.length ; i++) {
    alert("Элемент " + i + " равен:" + myArray[i] + ".");
}
```

Ограничивающее выражение оператора цикла использует оператор «меньше» (<), сравнивая индекс со свойством length массива Так как индексы отсчитываются с нуля, необходимо, чтобы максимальный индекс был на единицу меньше длины массива. Следовательно, нельзя использовать для этой цели оператор «меньше или равно» (<=). Если необходимо перебирать элементы массива в обратном порядке, в качестве начального значения для индекса следует использовать его размер, уменьшенный на 1:

```
var myArray = ["Alice", "Fred", "Jean", "Steve"];
for (var i = myArray.length  1; i >= 0 ; i-) {
    alert("Элемент " + i + " равен:" + myArray[i] + " ");
}
```

Если переменная цикла уже объявлена с помощью var, ее не нужно повторно объявлять в цикле.

Обсуждение

Нередко массив объектов (или коллекцию объектов DOM) перебирают для того, чтобы найти определенный элемент и использовать его индекс для дальнейших подстановок. Вот пример пары параллельных, но не связанных массивов, содержащих информацию соответственно об имени и возрасте людей:

```
var nameList = ["Alice", "Fred", "Jean", "Steve"];
var ageList = [23, 32, 28, 24];
```

Эти параллельные массивы можно применять в качестве таблиц подстановки. Показанная дальше функция получает имя человека и возвращает соответствующий ему возраст из другого массива:

```
function ageLookup(name) {
    for (var i = 0; i < nameList.length ; i++) {
        if (nameList[i] == name) {
            return ageList[i];
        }
    return "Невозможно найти " + name + " ";
!
```

Аналогично можно проверять свойства объектов в коллекции на соответствие какому-то критерию и использовать найденные индексы для изменения других свойств объектов. Следующая функция очищает содержимое всех текстовых полей на странице, даже если страница содержит несколько форм:

```
function clearTextBoxes() {
    var allInputs = document.getElementsByTagName("input");
    for (var i = 0; i < allInputs.length; i++) {
        if (allInputs[i].type == "text") {
```

```
            allInputs[i].value = "";
        }
    }
}
```

Для того чтобы добраться до каждого из элементов многомерного массива, следует взять многомерные (то есть вложенные) циклы. Например, следующий код, получив на входе двумерный массив, показанный в рецепте 3.2, переберет все элементы массива и вычислит их сумму:

```
var total = 0; // Сумма
for (var i = 0; i < salesArray.length; i++) {
    for (var j = 0; j < salesArray[i].length; j++) {
        total += salesArray[i][j];
    }
}
```

Вложенный цикл использует отдельную переменную-счетчик (j). Если представить двумерный массив в виде таблицы, как это сделано в рецепте 3.2, внешний цикл (с переменной i) будет перебирать строки, а вложенный (использующий переменную j) — столбцы. Следовательно, элементы массива будут перебираться в следующей последовательности:

```
ряд 0, ячейка 0
ряд 0, ячейка 1
ряд 0, ячейка 2
ряд 0, ячейка 3
ряд 1, ячейка 0
ряд 1, ячейка 1
```

Смотрите также

Рецепт 3.9 демонстрирует имитацию хэш-таблиц, являющихся скоростной альтернативой параллельных массивов. Рецепт 3.2 демонстрирует создание многомерных массивов.

3.5. Сортировка массива

NN2 **IE4**

Задача

Необходимо отсортировать массив строк или чисел.

Решение

Дчя того чтобы отсортировать массив от наименьших значений к наибольшим, можно применить метод sort() массива:

```
myArray.sort();
```

Этот метод меняет порядок элементов массива, который не может быть восстановлен, если сценарий не сохранил копию исходного массива. При сортировке многомерного массива сортируется только верхний уровень.

Обсуждение

Этот же метод можно использовать и для сортировки массивов строк, но сравнение строк при сортировке производится по ASCII-кодам их символов. Поэтому если строки массива имеют буквы в разном регистре, в полученном массиве сначала будут идти строки, начинающиеся с букв в нижнем регистре, потом — в верхнем (потому что ASCII-коды символов в верхнем регистре больше, чем коды символов в нижнем). Для более сложной сортировки следует определить функцию сравнения, которая будет вызываться из метода sort().

Функция сравнения является очень мощным инструментом управления данными в языке JavaScript. Для того чтобы применить ее при сортировке, необходимо передать ссылку на функцию как единственный параметр метода **sort()**.

При сортировке массива с привлечением такой функции интерпретатор многократно вызывает ее, передавая значения двух сравниваемых аргументов. Поэтому функция должна иметь два входных параметра. Задача функции — сравнить переданные ей значения и вернуть значение меньше нуля, нуль или больше нуля, в зависимости от отношения между значениями:

<0 второе значение должно быть расположено после первого;
О порядок этих двух значений не нужно менять;
>0 первое значение должно быть расположено после второго.

В качестве примера рассмотрим массив, состоящий из чисел. Если вызвать метод **sort()** без параметров, функция сравнения, принятая по умолчанию, будет сравнивать числа как строки, по их ASCII-кодам. Это может привести к тому, что число 10 окажется перед 4, потому что ASCII-код символа «1» меньше, чем код символа «4».

Для того чтобы отсортировать числовые значения в их естественном порядке, следует задать функцию сортировки, которая бы непосредственно сравнивала величины как числа:

```
function compareNumbers(a, b) {
    return a - b;
}
```

После этого можно вызвать сортировку так:

```
myArray.sort(compareNumbers);
```

За кулисами интерпретатор JavaScript передает этой функции пары сравниваемых значений. Если функция возвращает отрицательный результат, это означает, что данную пару значений следует поменять местами. После перебора всех пар значений массив будет отсортирован в нужном порядке. В качестве примера рассмотрим функцию, обеспечивающую сортировку массива чисел в порядке убывания:

```
function compareNumbers(a, b) {
    return b - a;
}
```

Пример более сложной сортировки, включая сортировку по длине строк в элементах массива, смотрите в рецепте 3.11.

Смотрите также

Рецепт 3.11 показывает, как сортировать массив объектов по значению определенного свойства этих объектов.

3.6. Объединение массивов

NN4 IE4

Задача

Необходимо смешать два или более массивов в один, большего размера.

Решение

Для объединения массивов применяйте метод массива concat(). Этот метод получает ссылку на второй массив в качестве своего аргумента:

```
var comboArray = myArray.concat(anoutherArray);
```

Массивы, объединяемые методом concat(), не меняются. Вместо этого метод возвращает новый массив, который можно сохранить в отдельной переменной. Исходный массив (тот, для которого вызывается метод) попадает в новый массив первым.

Для того чтобы объединить сразу несколько массивов, необходимо передать их имена как параметры метода concat(), разделяя запятой:

```
var comboArray = myArray.concat(otherArray1, otherArray3, otherArray3);
```

Элементы объединяемых массивов появляются в результирующем массиве в том же порядке, в каком массивы передавались методу.

Обсуждение

Функциональность метода concat() не ограничена слиянием нескольких массивов в один. Параметрами этого метода могут быть величины любого типа. При объединении такие значения просто копируются в результирующий массив, в том же порядке, в котором они перечислены в списке аргументов. Можно даже комбинировать массивы и другие типы данных в одном вызове метода.

Метод concat() дополняет четыре метода, позволяющих рассматривать массив как стек, добавляя и удаляя элементы по обоим концам массива. Метод push() добавляет один или несколько элементов в конец массива. Метод pop() извлекает из массива последний элемент и возвращает его значение. Для того чтобы производить те же операции с началом массива, можно использовать методы unshift() (добавление) и shift() (удаление). Все эти четыре метода реализованы в NN 4 и IE 5.5 для Windows.

Смотрите также

Рецепт 3.5 демонстрирует сортировку массива, которая может понадобиться после того, как к массиву добавлены новые элементы. Рецепт 3.7 показывает, как разделить массив на части.

3.7. Рассечение массива

NN4 **IE5.5(Win)**

Задача

Необходимо разделить один массив на два или больше сегментов.

Решение

Для того чтобы рассечь массив на части, примените к исходному массиву метод splice(). Этот метод доступен в NN 4 и IE 5.5 для Windows. Метод splice() требует два аргумента. Первый аргумент — индекс первого элемента вырезаемого фрагмента массива. Второй — количество элементов, которые будут вырезаны. Для примера рассмотрим такой массив:

```
var myArray = [10. 20. 30. 40. 50. 60. 70]:
```

Для того чтобы разбить его на два массива из трех и четырех элементов, сначала следует решить, какие элементы останутся в исходном массиве. В этом примере первые три элемента удаляются и помещаются в отдельный массив:

```
var subArray = myArray.splice(0.3):
```

После выполнения метода splice() массивы будут содержать следующие данные:

```
myArray = [40. 50. 60. 70]:
subArray = [10. 20. 30]:
```

Можно извлекать из массива любую непрерывную последовательность элементов. После выполнения метода исходный массив уменьшается, его размер становится равным числу оставшихся элементов. Оба массива никак не связаны друг с другом после разбиения.

Обсуждение

Метод splice() позволяет не только вырезать группу элементов из массива. Необязательные параметры, имеющиеся у этого метода, позволяют одной командой как удалять элементы, так и вставлять новые на их место. Более того, количество элементов, добавляемых в массив взамен, не обязательно должно совпадать с количеством удаляемых. Чтобы продемонстрировать это, начнем с простого массива:

```
var myArray = [10. 20. 30. 40. 50. 60. 70]:
```

Теперь удалим три элемента из середины и заменим их двумя новыми элементами:

```
var subArray = myArray.splice(2, 3, "Jane", "Joe");
```

После выполнения такого оператора массивы будут содержать такие данные:

```
myArray = [10, 20, "Joe", "Jane", 60, 70];
subArray = [30, 40, 50];
```

Использование splice() — один из лучших методов удаления элементов из массива. При таком использовании возвращаемое функцией значение можно игнорировать.

У метода **splice()** есть двойник, метод slice(). Он создает копию непрерывного участка исходного массива и помещает ее в новый массив. Различие между методами slice() и splice() состоит в том, что **sliceQ** не меняет исходный массив. Этот метод имеет два параметра, индексы первого и последнего элементов последовательности. Второй параметр можно опустить, при этом копируется фрагмент от указанного индекса до конца исходного массива.

Смотрите также

Рецепт 3.6 показывает, как объединить несколько массивов в один.

3.8. Создание пользовательского объекта

NN3 **IE4**

Задача

Необходимо создать объект для хранения вашей структуры данных.

Решение

Как и в случае с созданием массива, имеются полная и сокращенная форма создания объекта. Полная форма требует определить конструктор, который будет создавать новый объект, сокращенная форма использует специальные символы для разметки структуры объекта.

Конструктор похож на любую другую функцию JavaScript, но его назначение — определять начальную структуру объекта, имена его свойств и методов. Конструктор также может заполнять поля объекта начальными данными. Значения, записываемые в поля объекта при создании, обычно передаются через аргументы конструктора. Показанный ниже конструктор создает объект с двумя свойствами:

```
function coworker(name, age) {
    this.name = name;
    this.age = age;
}
```

Для того чтобы создавать объекты с помощью этого конструктора, следует применить ключевое слово new:

```
var empl - new coworker("Alice". 23):
var emp2 = new coworker("Fred". 32):
```

Ключевое слово **this,** используемое в конструкторе и в методах, указывает локальный контекст выполнения функции. Так как одна и та же функция служит для создания многих объектов, такое указание необходимо для привязки к конструируемому объекту.

Если вы предпочитаете не привлекать конструктор, можете создавать объекты с использованием сокращенного синтаксиса (доступен начиная с 4-й версии браузеров). При этом объект целиком описывается выражениями в фигурных скобках. В фигурные скобки помещаются пары, состоящие из имени и значения свойства. Имя от значения отделено двоеточием, а друг от друга пары разделены запятыми. Например, показанные выше два объекта с использованием сокращенного синтаксиса могут быть созданы так:

```
var empl = {name:"Alice", age:23}:
var emp2 - {name:"Fred". age:32}:
```

После создания объекта можно обращаться к его свойствам точно так же, как к свойствам любого другого объекта в JavaScript. Например, оператор, показывающий в диалоговом окне содержимое объекта **етр2,** может выглядеть так:

```
alert("Возраст сотрудника " + emp2.name + " " + emp2.age + " лет."):
```

Новые свойства можно добавлять к объекту и после того, как он создан, просто присваивая нужное значение свойству с любым именем. Например, для того чтобы добавить к объекту свойство, хранящее информацию о номере кабинета, можно ввести следующий код:

```
emp2.room = 320:
```

После такого присвоения у объекта **етр2,** и только у него, появится свойство **room** (дополнительные примеры смотрите в рецепте 3.12). Как видно из этого примера, свойство можно применять, не объявляя его в конструкторе. Это также значит, что можно быть весьма небрежным при создании объекта, формируя пустой объект, а затем помещая в него свойства одно за другим:

```
var empl - new Object();
empl.name - "Alice";
empl.age = 23:
```

Однако такой способ создания объектов сложен как при поддержке кода, так и при написании, если необходимо создать много объектов с одинаковой структурой.

Обсуждение

Выше было показано, как формировать свойства у объекта. Добавить к нему метод не намного сложнее. Для этого достаточно предварительно описать метод где-либо в коде как функцию, после чего присвоить ссылку на эту функцию методу объекта. Это можно сделать как в конструкторе, так и в фигурных скобках. Продолжая развитие только что описанных простых объектов, добавим

к ним метод, который показывает диалоговое окно с именем и возрастом сотрудника. Для начала напишем функцию, выполняющую это действие:

```
function showAll() {
    alert("Возраст сотрудника " + this.name +   ' + this.age + " лет ");
}
```

После чего добавим ссылку на эту функцию в методы объекта. Это можно сделать в конструкторе:

```
function coworker(name. age) {
    this.name = name;
    this.age = age:
    this.show - showAll;
}
```

Сокращенная форма создания подобного объекта будет выглядеть так:

```
var emp1 = {name:"Alice". age:23. show:showAll};
var emp2 = {name:"Fred". age:32. show showAll};
```

Теперь, для того чтобы выполнить метод, необходимо вызвать его у любого из объектов:

```
emp1.show();
```

Обратите внимание на то, как в метод передается контекст объекта, для которого он вызван. Ключевое слово this играет роль ссылки на объект и может быть использовано для получения доступа ко всем его свойствам и методам.

Реализация JavaScript в IE 4 и Navigator 4 предоставляет еще один оператор. Его можно применить в конструкторе для присвоения значений по умолчанию свойствам, не указанным в конструкторе при его вызове. Например, если в показанном выше примере опустить второй параметр при вызове конструктора объекта **coworker,** то в поле возраста будет помещено значение null. Такое поведение не всегда подходит, и более удобным было бы инициализировать это поле нулем. Это можно сделать, изменив конструктор следующим образом:

```
function coworker(name. age) f
    this.name = name:
    this.age = age || 0:
    this.show = showAll:
}
```

Здесь применен обыкновенный оператор JavaScript ИЛИ. Если первое значение, передаваемое этому оператору, равно null или не определено, в свойство записывается значение второго параметра. Такую конструкцию можно использовать в любом операторе присваивания, не только в конструкторе.

Одно из достоинств полной формы записи конструктора заключено в том, что можно вызывать методы объекта и функции еще при конструировании. Это может быть, например, процедура инициализации, которую нужно вызвать, как только объект был создан. Для того чтобы сделать это, просто добавьте вызов метода в конструктор. Показанный ниже конструктор создает уже известный нам объект coworker, но при этом показывает диалоговое окно с сообщением об объекте сразу после его создания. Диалоговое окно вызывает отдельная функция:

```
function verify(obj) {
    alert("Был создан объект " + obj.name + ".");
}
function coworker(name, age) {
    this.name = name;
    this.age = age;
    this.show = showAll;
    verify(this);
}
```

Если вызываемая из конструктора внешняя функция возвращает значение, оно может быть присвоено любому свойству этого объекта.

Если вы взяли на себя труд написания конструктора для сложной структуры данных, то, скорее всего, вы собираетесь использовать несколько подобных объектов. Но вместо того, чтобы позволять этим объектам в виде глобальных переменных свободно плавать в сценарии, гораздо удобнее хранить эти объекты в массиве. Как показано в рецепте 3.4, такая структура облегчает перебор коллекции похожих элементов. Например, можно использовать массив из объектов coworker для того, чтобы найти имена всех сотрудников, возраст которых удовлетворяет заданному критерию.

Создание массива объектов лишь немногим сложнее создания набора независимых объектов. Вот пример того, как сочетать вызов конструкторов объектов с конструктором массива:

```
var employeeDB = new Array();
employeeDB[employeeDB.length] = new coworker("Alice", 23);
employeeDB[employeeDB.length] = new coworker("Fred", 32);
employeeDB[employeeDB.length] = new coworker("Jean", 28);
employeeDB[employeeDB.length] = new coworker("Steve", 24);
```

То же самое можно проделать, используя сокращенный синтаксис:

```
var employeeDB = new Array();
employeeDB[employeeDB.length] = {name:"Alice", age:23, show:showAll};
employeeDB[employeeDB.length] = {name:"Fred", age:32, show:showAll};
employeeDB[employeeDB.length] = {name:"Jean", age:28, show:showAll};
employeeDB[employeeDB.length] = {name:"Steve", age:24, show:showAll};
```

Наконец, можно пойти по этому пути до конца, применяя сокращенную запись:

```
var employeeDB = [{name:"Alice", age:23, show:showAll},
                  {name:"Fred", age:32, show:showAll},
                  {name:"Jean", age:28, show:showAll},
                  {name:"Steve", age:24, show:showAll}];
```

Для того чтобы найти всех сотрудников, чей возраст лежит в определенных пределах, можно воспользоваться следующей функцией:

```
function findInAgeGroup(low, high) {
    var result = new Array();
    for (var i = 0; i < employeeDB.length; i++) {
        if (employeeDB[i].age >= low && employeeDB[i].age <= high) {
            result = result.concat(employeeDB[i].name);
        }
    }
    return result;
}
```

Она возвращает массив записей, для которых значение возраста лежит между величинами параметров функции low и high.

Массивы, как это показано в обсуждении рецептов 3.9 и 3.11, являются одной из наиболее гибких структур данных, доступных программисту на JavaScript. Поэтому при разработке страниц имеет смысл группировать похожие объекты в массив.

Смотрите также

Рецепт 3.9 показывает, как построить быструю хэш-таблицу на основе массива объектов. Рецепт 3.11 показывает сортировку массива объектов по значению определенного свойства.

3.9. Имитация хэш-таблицы

NN4 IE4

Задача

Необходимо обеспечить непосредственный доступ к элементам массива, минуя перебор всех элементов в поиске нужной записи. Особенно это важно в случае массива объектов или многомерного массива.

Решение

Используя тот факт, что массив в JavaScript является также объектом, можно добавлять этому объекту новые свойства, не затрагивая исходной его части. Значения таких свойств могут быть получены по их имени, как через строку (в этом случае имя свойства указывается в скобках, как индекс массива), так и обычным образом, применяя точку.

Ключевая идея реализации предлагаемой конструкции для уже существующего массива — создать для каждого элемента свойство с уникальным именем. Так, можно использовать значение имени объекта coworker в качестве имени свойства массива. Для этого после заполнения массива данными следует выполнить следующий оператор:

```
for (var i = 0; i < employeeDB.length; i++) {
    employeeDB[employeeDB[i].name] = employeeDB[i];
}
```

Для того чтобы определить возраст сотрудника без использования хэш-таблицы, необходимо сначала перебрать элементы массива и найти сотрудника с заданным именем. При использовании имитации хэш-таблицы достаточно просто обратиться к свойству для того, чтобы получить ссылку на нужный объект и свойство age:

```
var JeansAge = employeeDB["Jean"].age;
```

Скорее всего, для поиска придется воспользоваться строковой формой обращения к свойству, потому что информация для поиска приходит из текстового источника: поля ввода или элемента select.

Обсуждение

Сложно переоценить потребность в уникальных именах для свойств Пара совпадающих имен приведет к тому, что разные значения будут последовательно присвоены одному и тому же свойству, поэтому при поиске будет найден только второй из них.

Если одного свойства недостаточно для формирования уникального идентификатора, можно комбинировать несколько значений, чтобы добиться уникальности. Например, приведенную ниже таблицу можно превратить в соответствующий массив объектов.

Описание	1	2	3	4
Восточный (east)	2300	3105	2909	4800
Центральный (central)	1800	1940	2470	4350
Западный (west)	900	1200	1923	3810

Каждой клетке таблицы будет соответствовать отдельный объект, свойства которого содержат как само значение, так и информацию о его положении в таблице. Например:

```
var sales = new ArrayO:
sales[sales length] = {period:"1". region:"east". total:2300};
sales[sales length] = {period:"2". region:"east". total:3105}:

sales[sales length] = {period:"4". region:"west". total:3810};
```

Ни одно из свойств, по которым может потребоваться делать выборки из таблицы, не имеет уникальных значений. Четыре разных объекта имеют свойство region, равное west, и три — свойство period, равное 1. Но комбинация этих двух параметров образует уникальное значение для каждой клетки таблицы. Таким образом, используя имена вида region_period (например, central_2), сценарий может получить доступ к любой записи. Создание хэш-таблицы будет выглядеть так:

```
for (var I = 0: ш Б sales.length; i++) {
    sales[sales[i].region + sales[i].period] = sales[i]:
}
```

После этого, для доступа, например, к продажам третьей четверти центрального региона, можно использовать следующую запись:

```
sales["central_q3"].total
```

Когда в поле хэш-таблицы помещается ссылка на объект (как это и делалось в показанных выше примерах), дублирования данных не происходит. Записи хэш-таблицы просто указывают на исходные данные, поэтому любое изменение данных в массиве отразится и в таблице.

Каждый раз, когда вам необходим большой многомерный массив или коллекция объектов, в которой сценарий будет искать нужные объекты, старайтесь добавлять к массиву такие имитированные хэш-таблицы. Такой подход сочетает лучшие стороны обоих подходов. Применение массива позволяет перебирать элементы, если необходимо обработать каждую запись, и возможность быстрого доступа к любой записи без перебора.

Смотрите также

Рецепты 8.13, 10.8 и 15.4 содержат практические примеры использования хэш-таблиц.

3.10. Работа со свойствами объекта

NN2 IE3

Задача

Необходимо определить значение свойства объекта (или изменить его), но при этом как сам объект, так и его свойства могут меняться от проверки к проверке.

Решение

Для получения доступа ко всем свойствам объекта, независимо от их имени, можно использовать оператор цикла for/in. Показанная ниже функция выводит список всех свойств переданного ей объекта и их значения.

```
function listProperties(obj. objName) {
    var result = "":
    for (var i in obj) {
        result += objName + "." + i + "=" + obj[i] + "\n":

    alert(result);
}
```

Данный оператор цикла поочередно присваивает указанной переменной (в нашем случае это i) все имена всех свойств объекта (в строковой форме). Значения свойства можно определить, указав его имя в квадратных скобках после объекта (то есть obj[i] в этом примере).

Обсуждение

На рис. 3.1 показан результат выполнения этой функции для одного из объектов sales, использованного в рецепте 3.9.

Рис. 3.1. Пример перечисления свойств объекта

Показанный в функции listProperties() способ перебора свойств подходит не только для пользовательских объектов, но и для объектов DOM, однако результаты такого перебора для объектов DOM зависят от браузера. Так, IE для Windows перечисляет все обработчики событий объекта. Netscape 6, а также более поздние версии перечисляет как свойства, так и методы объекта. Opera, к сожалению, не позволяет перечислять свойства объектов DOM, но с пользовательскими объектами это работает.

Смотрите также

Рецепт 3.4 предлагает способ перебора всех записей в массиве.

3.11. Сортировка массива объектов

NN4 IE4

Задача

Необходимо отсортировать массив объектов, основываясь на значениях их свойств.

Решение

При сортировке массивов объектов используется логичное расширение функции сравнения, описанной для простых массивов в рецепте 3.5. Для этого определим функцию сравнения обычным образом, за исключением того, что сравниваться будут не сами объекты, а значения их свойств.

Чтобы продемонстрировать эту концепцию, начнем с массива объектов sales (величины продаж по регионам):

```
var sales = new Array();
sales[sales.length] = {period:"q1", region:"east", total:2300};
sales[sales.length] = {period:"q2", region:"east", total:3105};

sales[sales.length] = {period:"q4", region"west", total:3810};
```

Для того чтобы отсортировать такой массив в порядке убывания по величине продаж (total), функцию сравнения нужно определить следующим образом:

```
function compareTotals(a, b) {
    return b.total - a.total;
}
```

Для того чтобы отсортировать массив с использованием этой функции, следует передать ссылку на нее в метод sort().

```
sales.sort(compareTotals);
```

Вызов метода sort() изменяет порядок элементов в исходном массиве. Впоследствии можно вызвать другие методы сортировки (с другими функциями сравнения), чтобы сортировать элементы по другому критерию.

Обсуждение

Функции сравнения могут быть весьма сложными. Все зависит от структуры данных и необходимого порядка сортировки. Например, если объект содержит отдельные поля для месяца, дня и года, для сортировки массива таких объектов по датам можно использовать следующую функцию сравнения:

```
function compareDates(a, b) {
    var dateA = new Date(a.year, a.month, a.date);
    var dateB = new Date(b.year, b.month, b.date);
    return dateA   dateB;
}
```

При сортировке по значению строкового свойства можно исключить влияние регистра сравниваемых строк на их порядок. Для этого перед сравнением необходимо преобразовать строки целиком в верхний или целиком в нижний регистр. Следующая функция сортирует массив объектов по значению свойства **lastName**, игнорируя регистр букв:

```
function compareNames(a, b) {
    var nameA = a.lastName.toLowerCase();
    var nameB = b.lastName.toLowerCase();
    if (nameA < nameB) {return -1}
    if (nameA > nameB) {return 1}
    return 0;
}
```

Результат этой функции принимает одно из трех значений, описанных в рецепте 3.5. Поскольку метод **toLowerCase()** не меняет исходной строки, данные в массиве не искажаются при сортировке.

Смотрите также

Рецепт 3.5 содержит основные концепции сортировки массивов. Рецепт 14.16 показывает, как использовать сортировку массива объектов для сортировки данных в динамической HTML-таблице.

3.12. Изменение прототипа объекта

NN4 IE4

Задача

Необходимо добавить новое свойство или метод в объекты, которые уже созданы или которые только будут созданы.

Решение

Для того чтобы добавить свойство или метод ко всей группе объектов, созданных с помощью одного конструктора, следует присвоить имя и начальное значе-

ние свойства или метода свойству prototype объекта. Для демонстрации этой идеи начнем с конструктора объекта coworker и создадим четыре объекта, поместив их в массив.

```
function coworker(name. age) {
    this.name = name:
    this.age = age || 0:
    this.show = showAll;
}
var employeeDB = new ArrayO:
employeeDB[employeeDB.length] = new coworker("Alice". 23):
employeeDB[employeeDB.length] = new coworker("Fred". 32):
employeeDB[employeeDB.length] = new coworker("Jean". 28):
employeeDB[employeeDB.length] = new coworker("Steve". 24):
```

Каждый из этих объектов имеет один метод и два свойства. Каждый объект имеет собственные значения свойств, а название (и код) метода общие для всех объектов. При вызове метода объект, в контексте которого выполняется код, доступен через ключевое слово this.

Имеется возможность добавить новое свойство или метод прототипу объекта — абстракции, представляющей собой «лекало», по которому сделаны все объекты одного типа. Притом все объекты, даже те, которые были созданы раньше, получают это свойство или метод. Например, ко всем объектам, созданным конструктором coworker, можно добавить новое свойство, характеризующее занятость сотрудника. По умолчанию это свойство принимает значение "на работе":

```
coworker.prototype status = "на работе":
```

После выполнения такой команды каждый объект из массива emploeeDB получает свойство status, значение которого можно получить так:

```
employeeDB[i].status
```

Поначалу у всех объектов это свойство будет равно значению, заданному по умолчанию. Если же изменить его у одного из объектов, значение свойства изменится только у него, а остальные сохранят прежнее значение по умолчанию. Так, если выполнить команду:

```
employeeDB[2].status = "отпуск по болезни":
```

то для всех остальных объектов массива employeeDB значение свойства status сохранится.

Изменения значений свойств долговременны. Другими словами, если после всех этих изменений еще раз поменять значение свойства прототипа, то те объекты, у которых оно было изменено отдельно, сохранят свои значения. Для примера рассмотрим изменение, отражающее период отпусков для всех сотрудников компании:

```
coworker.prototype.status = "в отпуске":
```

После этого у всех объектов массива, кроме employeeDB[2], свойство status будет равно "в отпуске". Объект employeeDB[2] сохранит значение свойства равным строковому значению "в отпуске по болезни".

Для любого свойства объекта, унаследованного от свойства прототипа, всегда можно определить значение соответствующего свойства прототипа, даже если

само значение свойства было изменено. Вот пример команды, сравнивающей значение свойства status у объекта и у его прототипа:

```
if (employeeDB[2].status != employeeDB[2].constructor.prototype.status) {
    // Если эти два значения различны, то значение свойства было изменено.
}
```

Обращение к конструктору в JavaScript является аналогом обращения к super в некоторых других настоящих объектно-ориентированных языках.

Обсуждение

Последующее обсуждение предполагает, что у читателя есть опыт использования или понимание объектно-ориентированных концепций в языках наподобие Java. Но даже если у вас нет таких знаний, данная глава поможет вам увидеть некоторые наиболее запутанные и сложные возможности языка JavaScript.

Все объекты в JavaScript — пользовательские объекты, глобальные объекты языка и объекты DOM (в Netscape 6 и выше), используют наследование прототипов. Где бы ни встретилось обращение к свойству (или методу) объекта, JavaScript, чтобы определить его значение, следует по цепочке прототипов. Правила, которым следует интерпретатор, относительно просты:

- если свойству объекта присвоено собственное значение (эта ситуация встречается в 99 % случаев), это значение и рассматривается как значение свойства;
- если свойству объекта не присвоено никакого значения, интерпретатор ищет это свойство в прототипе конструктора данного объекта;
- если в прототипе конструктора нужное значение или свойство не найдено, интерпретатор следует по цепочке прототипов вплоть до базового объекта Object, если это необходимо;
- если свойство не найдено в цепочке прототипов, интерпретатор возвращает для этого свойства значение undefined;
- если цепь наследования прототипов содержит больше одного объекта, все они доступны из сценария, например конструктор объекта Array наследует конструктор базового объекта Object.

  Это означает, что свойство массива может быть определено в прототипе конструктора как объекта Object, так и объекта Array. Как было показано в секции «Решение», доступ к прототипу объекта Array можно получить следующим образом:

```
myArray.constructor.prototype.имяСвойства
```

Для того чтобы подняться на один уровень выше, к прототипу Object, следует использовать следующий код:

```
myArray.constructor.constructor.prototype.имяСвойства
```

В Navigator 4 и в более поздних версиях существует сокращенный способ отслеживания прототипов объектов: ключевое слово __proto__ (с двойным знаком подчеркивания до и после слова). Это ключевое слово упоминается здесь на тот

случай, если подобный синтаксис встретится вам при дальнейшем изучении объектно-ориентированных технологий в JavaScript. С использованием такой формы записи показанные выше примеры можно переписать таким образом:

```
myArray.__proto__.имяСвойства
myArray.__proto__.__proto__.имяСвойства
```

В других браузерах нет подобной сокращенной формы записи.

В JavaScript есть возможность имитации того, для чего в других языках используются конструкции **interface** и **implements** — это способ снабдить объект свойствами и методами другого объекта без создания дочернего класса разделяемого объекта. Реализация этого подхода не совсем интуитивно понятна, но, однажды освоенная, она неплохо работает.

Для демонстрации этой технологии начнем с уже знакомого объекта coworker, содержащего информацию о человеке. При создании объекта, описывающего члена команды разработчиков, можно заметить, что объект **coworker** уже содержит некоторые свойства, необходимые и в новом объекте. Тогда можно использовать два конструктора: один для объекта **coworker** и другой для teamMember (этот объект содержит информацию о члене команды разработчиков):

```
// Конструктор объекта coworker
function coworker(name, age) {
    this.name = name:
    this.age = age:
    this.show = showAll:
}
// конструктор объекта teamMember
function teamMember(name, age, projects, hours) {
    this projects = projects:
    this.hours = hours:
    this.member = coworker:
    this member(name, age):
}
```

Обратите внимание на то, что конструктор объекта **coworker** назначается методу this.member. Другими словами, это означает, что вызов метода member() у объекта **teamMember** приводит к созданию нового объекта **coworker.** Фактически следующий оператор конструктора вызывает этот метод, передавая ему часть своих аргументов. Создание набора объектов **teamMember** примет такой вид:

```
var projectTeams = new Array():
projectTeams[projectTeams.length] =
    new teamMember("Alice", 23, ["Gizmo"], 240):
projectTeams[projectTeams .length] =
    new teamMember("Fred", 32, ["Gizmo", "Widget"], 325).
projectTeams[projectTeams.length] =
    new teamMember("Jean", 28, ["Gizmo"], 200):
projectTeams[projectTeams.length] =
    new teamMember("Steve", 23, ["Widget"], 190):
```

Результатом такого смешивания конструкторов является то, что создаваемый объект будет иметь четыре свойства **(projects, hours, name, age)** и два метода

(showAll() и memberO). Метод **member()** вряд ли будет вызываться из сценария, поскольку он используется только внутри конструктора. Но все остальные части объекта **teamMember** вполне доступны и осмыслены.

Такая связь объектов приводит к неожиданному побочному эффекту, и конструктор coworker не входит в цепочку прототипов объекта teamMember. Таким образом, если добавить новое свойство прототипу конструктора coworker, ни один из объектов teamMember не получит это свойство.

Тем не менее есть способ поместить конструктор coworker в цепочку прототипов объекта teamMember. Для этого следует назначить в качестве прототипа teamMember пустой объект coworker:

```
teamMember.prototype = new coworker();
```

Это необходимо сделать до первого вызова конструктора teamMember, но зато можно отложить добавление методов и свойств в прототип объекта coworker. После такого назначения можно сделать что-нибудь подобное:

```
coworker.prototype.status = "на работе":
```

Теперь выполнение этой команды приведет к тому, что все объекты teamMember получат новое свойство status, по умолчанию равное "на работе". Конечно же, как и в случае прототипа любого другого свойства или метода, можно изменить значение этого свойства у отдельного объекта, не затрагивая все остальные.

В Navigator версии 6 и выше многие из этих возможностей языка применимы и к объектам DOM, наделяя их потенциально неограниченными возможностями. Эти браузеры дают сценарию доступ к конструкторам всех типов объектов DOM, тем самым позволяя добавлять свойства и методы к прототипам любых объектов DOM. Например, если нужно снабдить все таблицы (элементы table) методом, который удаляет все столбцы из вложенного элемента tbody (тело таблицы), сначала необходимо описать функцию, выполняющую это действие, а потом назначить это действие методом прототипа:

```
function clearTbody() {
    var tbodies = this.getElementsByTagName("tbody"):
    for (var i = 0: i < tbodies.length: i++) {
        while (tbodies[i].rows.length > 0) {
            tbodies[i].deleteRow(0);
        }
    }
}
HTMLTableElement.prototype.clear = clearTbody;
```

Впоследствии для любой таблицы можно вызывать любой метод clear() для того, чтобы удалить все ее строки:

```
document.getElementById("myTable").clear();
```

Браузеры, основанные на Mozilla, обеспечивают доступ из сценария ко всем типам объектов W3C DOM. Это позволяет реализовать любые, даже самые невероятные идеи о расширении свойств и методов любых элементов HTML и узлов текста. Можно начинать веселиться!

3.13. Преобразование массивов и объектов в строки

NN4 IE4

Задача

Необходимо преобразовать всю информацию массива или пользовательского объекта в строку для передачи через URL или сохранения в cookie. Также необходимо иметь возможность восстановить исходный объект или массив.

Решение

Описанную задачу можно решить с помощью библиотеки objectsArraysStrings.js, приведенной в обсуждении. Для того чтобы преобразовать пользовательский объект в строковую форму, необходимо вызвать функцию object2String(), передав ей в качестве аргумента ссылку на нужный объект:

```
var objAsString = object2String(myObj);
```

Аналогично, для преобразования в строковую форму массива служит функция array2String():

```
var arrayAsString = array2String(myArray);
```

Для обратного преобразования строк в исходные типы данных используются следующие функции библиотеки:

```
var myArray = string2Array(arrayAsString);
var myObj = string20bject(objString);
```

Обсуждение

Листинг 3.1 содержит код библиотеки objectsArraysStrings.js.

Листинг 3.1. Библиотека преобразований objectsArraysStrings.js

```
function object2String(obj) {
    var val, output = "";
    if (obj) {
        output += "{";
        for (var i in obj) {
            val = obj[i];
            switch (typeof val) {
                case ("object"):
                    if (val[0]) {
                        output += i + ":" + array2String(val) + ".";
                    } else {
                        output += i + ":" + object2String(val) + ".";
                    }
                    break;
                case ("string"):
```

продолжение

Листинг 3.1 (продолжение)

```
                output += i + ":'" + escape(val) + "'.";
                break;
            default:
                output += i + ":" + val + ".";
        }
    }
    output = output.substring(0, output.length-1) + "}";
}
return output;
}

function array2String(array) {
    var output = "";
    if (array) {
        output += "[";
        for (var i in array) {
            val = array[i];
            switch (typeof val) {
                case ("object"):
                    if (val[0]) {
                        output += array2String(val) + ".";
                    } else {
                        output += object2String(val) + ".";
                    }
                    break;
                case ("string"):
                    output += " " + escape(val) + "'.";
                    break;
                default:
                    output += val + ".";
            }
        }
        output = output.substring(0, output.length-1) + "]";
    }
    return output;
}

function string2Object(string) {
    eval("var result = " + string);
    return result;
}

function string2Array(string) {
    eval("var result = " + string);
    return result;
}
```

Первые две функции библиотеки производят преобразование данных в строки. Первая, object2String(), перерабатывает все свойства пользовательского объекта и формирует строку в том же формате, который используется при сокращенной записи конструктора объекта с помощью фигурных скобок. Единственный параметр этой функции — ссылка на преобразуемый объект. Возвращаемое зна-

чение представляет собой строку, включающую в себя описание объекта и фигурные скобки.

Для демонстрации работы функции object2String() рассмотрим конструктор простого объекта:

```
function coworker(name. age. dept) {
    this.name = name;
    this.age = age:
    this.department = dept:
}
```

Создадим новый объект:

```
var kevin = new coworker ("Кевин". 28. "Бухгалтерский отдел"):
```

Преобразуем объект в строку:

```
var objStr = object2String(kevin):
```

Строка, соответствующая этому объекту, равна:

```
{name:"Кевин",age:28,department:'Бухгалтерский%20отдел'}
```

Вторая из библиотечных функций, array2String(), преобразует массив в строковую запись с использованием квадратных скобок. Обе функции иногда используют друг друга. Например, при преобразовании в строку объекта, одно из свойств которого является массивом, это свойство преобразуется с помощью метода array2String(). Аналогично, преобразование массива объектов в строку требует использования функции object2String() для преобразования отдельных объектов.

Для восстановления объекта или массива из строки служит одна из двух оставшихся функций. Обе эти функции выполняют одинаковые действия, разные имена используются только для улучшения читаемости кода. Несмотря на предупреждения, которые вы можете увидеть в этой книге о низкой производительности кода, использующего функцию eval(). она здесь необходима.

То, какая из двух функций преобразования в строку должна применяться, определяет тип внешнего объекта. Даже если объекты играют в структуре сценария большую роль, чем массивы, массив может служить контейнером для таких объектов, как это показано в рецепте 3.8. В этом случае для преобразования всего массива в строку следует взять функцию array2String(), которая самостоятельно преобразует в строковое представление отдельные объекты.

Смотрите также

Рецепты 3.1 и 3.8 демонстрируют сокращенный способ создания объектов и массивов, имитируемый в этом рецепте. Рецепт 4.8 содержит обсуждение функции eval(). В рецепте 10.6 приводится действующий пример, использующий эту библиотеку для передачи объектов между страницами с помощью URL.

4 Переменные, функции и управление последовательностью выполнения

4.0. Вступление

В этой главе рассматривается несколько центральных понятий JavaScript. Несколько их этих рецептов (или ваших собственных их вариаций) могут стать обязательной частью ежедневного меню. Но даже если вы редко применяете эти конструкции, глава поможет вам освежить память и быстро сориентироваться, если они вдруг понадобятся.

Используя даже такие простые объекты, как переменные и функции, JavaScript имеет множество нюансов и особенностей, которые легко забыть. К тому же, ряд таких концепций, как обработка исключений и оператор **try/catch,** сравнительно новы для современных браузеров. Авторы сценариев, не имеющие достаточно опыта в программировании, нередко весьма небрежно относятся к обработке ошибок, что может иметь последствия в будущем. С другой стороны, варианты реализации обработки ошибок в разных браузерах зачастую несовместимы. Но даже если вы еще не используете обработку исключений в своих сценариях (возможно, из-за необходимости обеспечивать обратную совместимость), быть знакомым с этой концепцией необходимо. Со временем, когда рекомендации W3C DOM будут полностью реализованы в браузерах, понятие «безопасного сценария» обязательно будет включать в себя регулярное применение обработки исключений.

В завершении этой главы есть несколько советов по поводу повышения производительности сценариев. Маленькие сценарии могут быть неэффективными, но для больших проектов, работающих со значительными количествами скрытых от пользователя данных, это недопустимо. Здесь вы узнаете несколько правил, которые впоследствии можно применять и в небольших сценариях.

4.1. Создание переменной

NN2 IE3

Задача

Необходимо создать и использовать переменную, локальную или глобальную.

Решение

Для описания переменной служит ключевое слово **var** Значение переменной можно присвоить непосредственно на месте ее объявления или позже. Если переменная объявляется вне тела функции, она считается глобальной:

```
var myVar = someValue;
```

Это приводит к тому, что все операторы на странице, включая те, что принадлежат функциям, получают возможность считывать и изменять значение такой переменной.

Если же объявить переменную внутри тела функции, к такой переменной смогут обращаться только операторы этой функции:

```
function myFunction() {
    var myFuncVar = someValue:

}
```

Операторы, расположенные вне функции **myFunction(),** не смогут получить доступ к этой переменной, область видимости которой ограничена одной функцией.

Обсуждение

У переменных в JavaScript нет никаких существенных ограничений на объем данных, которые они могут содержать. Максимальный объем определен исключительно объемом оперативной памяти, доступной браузеру, этот параметр недоступен для сценария.

Понятие области видимости является одним из важнейших для понимания в JavaScript. Глобальные переменные доступны не только из того окна или фрейма, где они были объявлены. Можно получить доступ к ним и из других окон и фреймов (если они были доставлены с того же домена и сервера), используя ссылку на нужное окно. Например, оператор из фрейма, содержащего меню, может обратиться к глобальной переменой **myVar** во фрейме content таким образом:

```
parent.content.myVar
```

Не нужно беспокоиться о совпадающих именах глобальных переменных в разных окнах и фреймах, потому что ссылки на такие переменные всегда будут записывать по-разному.

Следует быть внимательнее, объявляя с помощью ключевого слова var переменные в теле функции. Если забыть указать при объявлении ключевое слово var, переменная будет рассматриваться как глобальная. Поэтому если существует глобальная переменная с данным именем, оператор присваивания изменит ее значение, вместо того чтобы использовать локальную. Самый надежный способ обойти проблемы такого рода — всегда применять ключевое слово **var** при объявлении переменной, независимо от того, где находится переменная. Несмотря на то что это ключевое слово необязательно для глобальных переменных, его использование считается хорошим тоном в кодировании. Такая запись позволяет легко найти место, где переменная впервые используется в сценарии.

Несмотря на то что в некоторых языках программирования проводится различие между объявлением переменной (фактически, это выделение памяти под переменную) и ее инициализацией (запись в нее значения), JavaScript избавляет программиста от необходимости заботиться о выделении памяти. В JavaScript переменные по-настоящему переменны, поскольку могут менять не только свое значение, но и тип данных (это просто следствие нестрогой типизации JavaScript).

Продолжая разговор о стиле программирования, можно посоветовать объявлять все глобальные переменные в начале сценария, точно так же часто используемые локальные переменные лучше объявлять в начале кода функции. Даже если на данном этапе не существует значения, которое можно было бы в нее поместить, можно просто объявить переменную оператором наподобие приведенного ниже:

```
var myVar;
```

Если нужно объявить много переменных, можно применить следующую компактную запись, разделяя имена переменных запятыми:

```
var myVar, counter, fred, i, j;
```

В одном операторе можно даже совмещать простые объявления и объявления с инициализацией:

```
var myVar, counter = 0, fred, i, j;
```

В примерах, которые можно встретить на страницах этой книги, переменные будут объявляться и инициализироваться в начале сценария, но не всегда. Нередко переменные, которые будут использоваться как счетчики в цикле for, объявляются (с помощью ключевого слова var) непосредственно перед циклом. Например, если потребуется использовать пару вложенных циклов, можно объявить переменные непосредственно перед внешним оператором цикла:

```
var i, j;
for (i = 0; i < array1.length; i++) {
    for (j = 0; j < array1[i].array2.length; j++) {

    }
}
```

Это не более чем стиль, принятый в этой книге. Но есть и другая причина объявлять переменные вне тела цикла. Если использовать ключевое слово var непосредственно при инициализации счетчика цикла (например, так: var j = 0;), оператор var во вложенном цикле будет выполняться при каждой итерации внешнего цикла. На внутреннем уровне интерпретатор JavaScript при каждом

использовании ключевого слова var выделяет новую переменную. К счастью, интерпретатор отслеживает повторно объявляемые переменные, но все равно это приводит к ненужному расходованию ресурсов. Лучше всего однократно объявить и инициализировать переменную, после чего изменять ее значение сколь угодно часто. Так, в сложных функциях, содержащих несколько циклов, можно объявить переменную-счетчик однократно, в начале функции, после чего просто присваивать ей начальное значение в начале каждого нового цикла.

Что касается имен переменных, то есть несколько явных правил и подразумевающихся традиций, которые можно использовать. Из них правила более важны. Имя переменной не должно:

- начинаться с числа;
- содержать пробелы или другие пробельные символы;
- содержать знаки препинания или специальные символы, кроме знака подчеркивания;
- быть заключено в кавычки;
- быть зарезервированным словом стандарта ECMAScript (см. приложение B).

Соглашения между программистами о том, какие имена давать переменным, не являются строгими и не влияют на работоспособность сценария. Тем не менее они улучшают читаемость кода и помогают, когда нужно вспомнить, что должен делать код, написанный шесть месяцев назад.

Основное назначение правил именования — помочь определить, какие данные содержит переменная с заданным именем (на практике имена часто называют идентификаторами). В сценарии, замусоренном множеством одно- и двухбуквенных переменных, довольно сложно отслеживать сами значения и логику работы программы. С другой стороны, по соображениям производительности не стоит позволять именам быть чрезмерно длинными. Чем имя короче, тем лучше, до тех пор пока оно не превращается в бессмысленные значки. Если для описания назначения переменной нужно два или больше слов, можно объединить их в одно с помощью символа подчеркивания, или сделав первую букву каждого нового слова заглавной. Последний способ применяется в этой книге повсеместно. Вот примеры правильно описанных переменных:

```
var teamMember = "Джордж";
var team_member = "Джордж";
```

Те же правила можно применять и при задании атрибутов элемента name и id. Тогда не возникнет сложностей при использовании таких идентификаторов для доступа к объектам DOM.

Имена переменных чувствительны к регистру. Таким образом, возможно (но, опять же, не всегда рекомендуется) применять для разных значений идентификаторы, различающиеся только регистром букв. Часто принято выделять переменные, значения которых не будут меняться на всем протяжении сценария (то есть константы) и формировать для них имена целиком в верхнем регистре. В Netscape 6 выше реализовано ожидающее своего включения в стандарт ECMAScript ключевое слово const, которое нужно использовать вместо var для описания константы. Ни один другой браузер пока не поддерживает это ключевое слово, поэтому вместо констант можно брать обычные переменные, стараясь не изменять их.

JavaScript в зависимости от типа данных может помещать в переменную данные как «по значению», так и «по ссылке». Если данные представляют собой настоящий объект любого рода (например, объект DOM, массив или пользовательский объект), то переменная содержит ссылку на этот объект. Такие переменные можно применять как замену ссылки на объект:

```
var elem = document.getElementById("myTable");
var padWidth = elem.cellPadding
```

Но если данные представляют собой простое значение (строку, число или булевское значение), переменная содержит копию этих данных, никак не связанную с объектом, из которого эти данные пришли. Таким образом, переменная padWidth в показанном выше примере хранит простое строковое значение. Если присвоить ей новое значение, это никак не повлияет на элемент table. Чтоб изменить свойство, придется вернуться к ссылке на объект и использовать ее для изменения свойства:

```
elem.cellPadding = "10";
```

Если же свойство объекта само является объектом, после присваивания переменная сохраняет с ним связь:

```
var elem = document.getElementById("myTable");
var elemStyle = elem.style;
elemstyle.fontSize = "18px";
```

Будьте осторожны при присваивании объектов DOM переменным. Может показаться, что такие переменные всего лишь копии объектов, но изменения, внесенные в такую переменную, влияют на элемент документа.

Смотрите также

В главах 1—3 обсуждается присваивание переменным значений разных типов: строк, чисел, массивов и объектов. Рецепт 4.8 объясняет влияние длины имени переменной на производительность.

4.2. Функции

NN2 **IE3**

Задача

Необходимо задать функцию, которую можно использовать в любом месте страницы.

Решение

Объявление функции, которой не передается параметров, имеет следующий формат:

```
function имяФункции() {
    // Код функции
```

Если же функция должна получать параметры из вызывающего оператора, необходимо в скобках перечислить имена переменных для этих параметров:

```
function имяФункции(аргумент1, аргумент2[,     аргументN]) {
    // Код функции
}
```

Можно описывать столько параметров, сколько нужно. Переменные, описанные в параметрах, становятся локальными переменными для этой функции (оператор var здесь не нужен). Вследствие нестрогой типизации, принятой в JavaScript, параметры могут содержать данные любого типа.

Фигурные скобки, обрамляющие тело функции, необходимы только в том случае, если ее код содержит два или более операторов. Тем не менее хорошим тоном считается использование фигурных скобок во всех случаях, даже когда она состоит из одного оператора. Это соглашение улучшает читаемость кода и применяется во всех примерах этой книги.

Большая часть сложных сценариев в этой книге используют функции, некоторые с параметрами, некоторые — без. В изобилии имеются и практические примеры, особенно это касается рецептов, содержащих внешние библиотеки, такие как библиотека DHTML API в рецепте 13.3.

Обсуждение

В JavaScript функция является объектом, и имя, которое ей присвоено, является ее идентификатором (а значит, чувствительно к регистру букв). Поэтому нельзя брать в качестве имени функции зарезервированные слова JavaScript, а также имена других глобальных объектов, например переменных или идентификаторов объектов (в IE). Если в сценарии описано две функции с одинаковым именем, применяется только та из них, которая описана последней. В JavaScript нет понятия перегрузки функций или методов, которое есть в таких языках, как Java (при использовании перегрузки методы, имеющие одинаковое имя, но разное число аргументов, выступают как различные).

Для вызова функции служат скобки:

```
myFunc();
myFunc2("hello", 42);
```

Иногда возникает необходимость присвоить свойству ссылку на функцию. Например, при назначении элементу обработчика события в свойство помещается ссылка на функцию (см. главу 9). Ссылка на функцию представляет собой имя функции, записанное без скобок, кавычек или параметров:

```
document.onclick = myFunc;
```

Такого вида присваивание только подготавливает возможность для последующего вызова функции.

В некоторых языках программирования различаются исполняемые блоки, которые выполняются сами по себе, и блоки, возвращающие значение. В JavaScript имеется только один тип функций. Если функция содержит оператор return, она возвращает значение, в противном случае никакого значения не возвращается.

Функции, выполняющие то, что в других языках обычно выполняют процедуры, зачастую возвращают значение только для того, чтобы передать в вызывающий оператор код завершения. Возвращаемое значение можно присвоить переменной или использовать в вызове другой функции или метода. Рецепт 15.6 демонстрирует такую возможность. Этот рецепт показывает, как вывести при загрузке время суток (утро, день или вечер) в приветствии на странице. Строку, содержащую время суток, формирует описанная в начале страницы функция getDayPart():

```
function getDayPartO {
    var oneDate = new Date();
    var theHour = oneDate.getHours();
    if (theHour < 12) {
        return "утро":
    } else if (theHour < 18) {
        return "вечером"-
    }else {
        return "днем":
    }
}
```

Эта функция вызывается как параметр метода document.write(), помещающего текст в страницу:

```
<script language="JavaScript" type="text/javascript">
<!-
document.write("С добрым " + getDayPartO + " и добро пожаловать")
//->
</script>
<noscript>Добро пожаловать</noscript>
 в GinatCo.
```

Не обязательно передавать в функцию именно то число аргументов, которое указано при ее задании. Если функция из разных мест сценария вызывается с разным числом аргументов, в функции можно получить значения аргументов, используя свойство arguments вместо имен аргументов:

```
function myFunc() {
    for (var i = 0: i < myFunc.arguments.length: i++) {
        // каждый элемент массива arguments представляет собой один
        // аргумент. Они следуют в том порядке, в котором переданы функции
    }
}
```

Большинство функций (за исключением, например, вложенных функций, описанных в рецепте 4.3) являются глобальными объектами в контексте окна, содержащего текущую страницу. Как и к глобальным переменным, к этим функциям можно обратиться и из других окон и фреймов. Раздел «Фреймы как окна» в главе 7 содержит примеры обращения к содержимому других фреймов.

Насколько большой должна быть отдельная функция — вопрос стиля. Для упрощения отладки и поддержки может быть удобным разбить код большой функции на секции и выделить их в отдельные функции. Если одна и та же последовательность операторов встречается в коде несколько раз, эти операторы являются превосходными кандидатами на исключение их в отдельную функцию.

Необходимо также решить, где ставить фигурные скобки. В этой книге принято соглашение ставить открывающую скобку в той же строке, где и имя функции, а закрывающую — с тем же отступом, что и имя функции. Но можно ставить открывающую скобку и ниже имени функции:

```
function myFunc()
{
    // Код функции
}
```

Некоторые считают, что при таком оформлении проще определять пары соответствующих скобок. Очень короткие функции можно записывать в одну строчку:

```
function myFunc0 {// Код функции }
```

Вы можете менять стиль, главное, чтобы он оставался логичным и удобным для восприятия.

Смотрите также

Рецепт 4.1 содержит обсуждение о хранении данных в переменных «по ссылке» и «по значению». Это обсуждение в равной мере относится как к переменным, так и к аргументам функции. Рецепт 4.3 демонстрирует вложение функций.

4.3. Вложение функций

NN4 IE4

Задача

Необходимо создать функцию, принадлежащую другой функции (и доступную только из нее).

Решение

Начиная с 4-й версии браузеров IE и Netscape, можно помещать описание одной функции внутрь другой, используя для этого следующий синтаксис:

```
function myFuncA() {
    // операторы функции myFuncA0
    function myFuncB() {
        // Операторы вложенной функции myFuncB0
    }
    // операторы функции myFuncA0
```

При такой записи к вложенной функции могут обращаться только операторы внешней (в случае Netscape 6 и более поздних версий имеются исключения, смотрите обсуждение). Операторы вложенной функции имеют доступ к переменным внешней, как к глобальным. Тем не менее внешняя функция не может обращаться к переменным вложенной.

Обсуждение

Основная идея использования вложенных функций — это возможность объединить все действия, относящиеся ко внешней функции, при этом оставляя их приватными по отношению к ней. Благодаря тому, что вложенные функции не попадают в глобальное пространство имен, можно повторно использовать их имена в качестве имен глобальных объектов или функций, вложенных в другую функцию.

В Netscape 6 и более поздних версиях понятие вложенной функции расширяется удобным и логичным образом. При этом вложенная функция является методом внешней. Есть несколько интересных вариантов реализации этой идеи. Рассмотрим такой набор функций:

```
function myFuncA() {
    var valueA = "A":
    myFuncB():
    function myFuncB() {
        var valueB = "B":
        alert(valueB);
        alert(valueA):
    }
}
```

Если из глобального пространства вызвать функцию **myFuncA()**, она вызовет вложенную функцию, а та в свою очередь покажет значения двух переменных, описанных во внешней и вложенной функциях. IE ведет себя именно таким образом, но в Netscape 6 можно вызвать вложенную функцию сразу, минуя вызов внешней:

```
myFuncA.myFuncB();
```

Так как внешняя функция при этом не выполняется, переменная A остается неинициализированной. Поэтому при запуске myFuncB() отображается только значение переменной B, переменная A отображается как undefined. Таким образом, получить доступ к переменным внешней функции из внутренней можно только в том случае, если внешняя функция выполняется.

Смотрите также

В рецепте 4.1 обсуждается понятие области видимости переменных.

4.4. Создание безымянной функции

NN4 IE4

Задача

Необходимо создать функцию в виде выражения, которую можно, например, передать в виде параметра конструктору или назначить методу объекта.

Решение

Можно использовать альтернативную форму записи, не определяя именованную функцию (как это делается в рецепте 4.2). Такой синтаксис, называемый безымянной функцией, содержит в себе все компоненты описания, кроме имени функции. Запись приведена ниже:

```
var someReference = function() { Операторы функции };
```

Операторы в фигурных скобках представляют собой обычные операторы JavaScript, разделенные точкой с запятой. Можно также определить аргументы функции, если это необходимо:

```
var someReference = function(параметр1[....параметрN]) {
    Операторы функции };
```

Вызвать функцию, заданную таким образом, можно, используя ссылку на нее:

```
someReference();
```

Обсуждение

При создании безымянной функции формируется объект типа function. Таким образом, в правой части оператора присваивания может стоять любое выражение, где требуется ссылка на функцию (то есть ее имя без скобок). В качестве примера рассмотрим сокращенную форму записи конструктора объекта из рецепта 3.8. В том рецепте сначала описывалась функция, которая затем назначалась методом для четырех создаваемых объектов:

```
function showAll() {
    alert("Возраст сотрудника " + this.name + " " + this.age + " лет.");
}
var employeeDB = [{name:"Alice", age:23, show:showAll},
                  {name:"Fred", age:32, show:showAll},
                  {name:"Jean", age:28, show:showAll},
                  {name:"Steve", age:24, show:showAll}];
```

Обратите внимание на то, как ссылка на функцию showAll() назначается в качестве метода show создаваемым объектам. Впоследствии этот метод можно вызвать следующим образом:

```
employeeDB[2].show();
```

Теперь в качестве примера назначим методом show для первого объекта безымянную функцию вместо ссылки на showAll():

```
var employeeDB = [{name:"Alice", age:23,
                   show:function() {
                       alert("Доступ к возрасту Alice закрыт")}},
                  {name:"Fred", age:32, show:showAll},
                  {name:"Jean", age:28, show:showAll},
                  {name:"Steve", age:24, show:showAll}];
```

Если теперь вызвать метод employeeDB[0].show(), появится специальное сообщение, потому что вместо showAll() выполняется код безымянной функции.

Теперь создавать внешнюю функцию с собственным идентификатором нужно лишь как начальный шаг описания метода.

Смотрите также

Рецепт 4.2 показывает, как создавать обыкновенные именованные функции.

4.5. Отложенный вызов функции

NN2 **IE3**

Задача

Необходимо запустить функцию в заданный момент в ближайшем будущем.

Решение

Метод window.setTimeout() однократно вызывает функцию после истечения заданного интервала в миллисекундах. Фактически при этом устанавливается таймер, запускающим при срабатывании выбранную функцию. Ссылка на функцию должна быть передана в виде строки со скобками, как в показанном ниже примере:

```
var timeoutID = setTimeout "myFunc()", 5000);
```

Данный метод возвращает идентификатор операции отложенного вызова, который следует сохранить в переменной. Потом можно отменить срабатывание таймера, вызвав метод clearTimeout(), передав ему идентификатор вызова:

```
clearTimeout(timeoutID );
```

После того как таймер установлен, оставшаяся часть сценария продолжает выполняться как обычно. Так что было бы неплохой идеей ставить вызов setTimeout() последним оператором функции.

Обсуждение

Важно понимать, чего не делает метод setTimeout(). Он не прерывает процесс вычислений и не создает задержки выполнения основного кода. Вместо этого он взводит внутренний таймер, который при срабатывании вызывает указанную функцию. Например, при создании слайд-шоу, когда браузер должен перемешаться на следующую страницу каждые 15 секунд при отсутствии действий со стороны пользователя, можно установить таймер в обработчике события onload. Также нужно определить функцию resetTimer(), обнуляющую значение таймера.

```
var timeoutID:
function goNextPage() {
    location.href = "slide3.html";
}
function resetTimer() {
```

```
    clearTimeout(timeoutID);
    timeoutID = setTimeout("goNextPage()"  15000);
}
```

Кроме того, следует установить обработчик, например, события onmouseover для того, чтобы сбрасывать таймер при каждом движении мыши:

```
window.onmousemove = resetTimer;
```

Функция resetTimer() автоматически отменяет предыдущий вызов функции и устанавливает новый 15-секундный таймер.

Если вызываемая функция должна получать параметры, можно включить их в строку ее вызова, даже если эти параметры представляют собой переменные. Но важно, чтобы эти переменные не являлись ссылками на объекты. Параметры должны находиться в таком виде, чтобы выдержать преобразование в строку, требуемую методом setTimeout(). Рецепт 8.4 демонстрирует, как передать имена относящихся к странице объектов DOM в метод setTimeout(). Самое сложное в этом примере — соблюдение нужного порядка кавычек:

```
function isEMailAddr(elem) {
    var str = elem.value;
    var re = /^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]{2,7}$/;
    if (!str.match(re)) {
        alert("Проверка правильности формата e-mail адреса.");
        setTimeout("focusElement('" + elem.form.name +
            "', '" + elem.name + "')", 0);
        return false;
    } else {
        return true;
```

В этом примере функция focusElement() должна получать два параметра, обозначающие как имя формы, так и имя текстового поля. Оба параметра метода focusElement() представляют собой строки, а поскольку первый аргумент метода setTimeout() также является строкой, то необходимо форсировать «строчность» параметров метода focusElement() с помощью кавычек. (Величина задержки в 0 миллисекунд в этом примере — не ошибка. Объяснение дается в рецепте 8.4.)

Если вы ищете способ обеспечить задержку между отдельными операторами функции, JavaScript не сможет предложить ничего из того, что имеется в других языках программирования. Нужного результата можно добиться, разбив функцию на несколько отдельных функций, по одной на каждую секцию кода, выполняемую между задержками. Отдельные блоки можно связать друг с другом, завершая их вызовом метода setTimeoutO, запускающего следующую функцию через нужное количество времени:

```
function mainFunc() {
    // Начальная последовательность операторов
    setTimeout("funcPart2()"   10000);
}
function funcPart2() {
    // промежуточные операторы
    setTimeout("finishFunc()", 5000);
```

```
function finishFunc() {
    // Завершающие операторы
}
```

Связанные таким образом функции не обязательно должны быть расположены рядом. Если необходимо, чтобы эти функции работали с одними и теми же данными, нужные данные можно передавать от одной из них к другой через параметры (в этом случае они не должны быть объектами) или можно сделать их глобальными. Если эти значения связаны друг с другом, имеет смысл объединить их в один пользовательский объект. Это дает возможность оценить действия, выполняемые каждым из сегментов.

Есть еще один метод JavaScript, setInterval(), который во многом похож на setTimeout(), но вызывает указанную функцию не однократно, а до тех пор, пока ему не будет указано прекратить работу (с помощью метода clearInterval()). Второй параметр этого метода указывает интервал между вызовами.

Смотрите также

Рецепт 8.4 демонстрирует использование метода setTimeout() для синхронизации выполнения сценария. В рецепте 10.11 показывается последовательность из трех функций инициализации, связанных друг с другом с помощью вызовов setTimeout() для поддержания синхронизации. Рецепт 12.3 предлагает пример выполнения функции фиксированное число раз с помощью счетчика в вызываемой функции. Рецепты 13.9 и 13.10 демонстрируют применение метода setInterval() в анимации.

4.6. Условное ветвление выполнения

NN4 **IE4**

Задача

Необходимо выполнять различные участки кода в зависимости от внешних значений, таких как булевские переменные, данные в полях ввода или значения элементов select.

Решение

Для управления исполнением сценария используются конструкции if, if/else или switch. Если необходимо выполнить секцию кода только в том случае, если выполняется некоторое условие, следует использовать оператор if с условным выражением, проверяющим правильность условия:

```
if (условие) {
    // Операторы, выполняемые, если условие = true
```

Чтобы выполнить одну ветвь кода при выполнении условия и другую, если оно не выполняется, следует использовать конструкцию if/else:

```
if (условие) {
    // Операторы, выполняемые, если условие = true
} else {
    // Операторы, выполняемые, если условие = false
}
```

Можно уточнить пункт else, используя дополнительные проверки условий:

```
if (условие1) {
    // Операторы, выполняемые, если условие1 = true
} else if (условие2) {
    // Операторы, выполняемые, если условие1 = false и условие2 истинно
} else {
    // Операторы, выполняемые, если условие1 = false и условие2 = false
}
```

Для проверки многозначных условий, имеющих числовое или строковое значение, служит оператор switch:

```
switch (выражение) {
    case значение1:
        // Операторы, выполняемые, если выражение = значение1
        break;
    case значение2:
        // Операторы, выполняемые, если выражение = значение2
        break;

    default:
        // Операторы, выполняемые, если выражение не равно ни одному
        // из перечисленных значений
        break;
}
```

Ключевое слово break в каждой из ветвей case обеспечивает то, что необязательная ветвь default не будет выполнена.

Обсуждение

Условные выражения в операторах if и if/else должны представлять собой булевские значения, равные true или false. Обычно в таких выражениях используются операторы сравнения (==, ===, !=, !==, <, <=, >, >=) для определения отношения двух значений. В большинстве случаев значение переменной сравнивается с константой или известным значением:

```
var theMonth = myDateObj.getMonth();
if (theMonth == 1) {
    // это значение означает месяц февраль
    monthLength = getLeapMonthLength(myDateObj);
} else {
    monthLength = getMonthLength(myDateObj);
```

В JavaScript имеется несколько сокращений для записи условных выражений. Эти сокращения удобны в том случае, когда необходимо проверять наличие некоторого объекта или свойства. В табл. 4.1 перечислены значения, которые автоматически преобразуются в true или false при подстановке их на место условного выражения. Например, существующий объект преобразуется в значение true, что позволяет использовать подобные конструкции:

```
if (myObj) {
    // объект myObj существует, используем его
}
```

Таблица 4.1. Эквиваленты условных выражений

true (истина)	false (ложь)
Строка, содержащая не менее одного символа	Пустая строка
Число, не равное нулю	0
Значение, не равное null	null
Ссылка на существующий объект	Ссылка на несуществующий объект
Свойство объекта существует и равно непустой строке или ненулевому числу	Несуществующее свойство объекта или свойство, равное пустой строке или 0

При проверке существования свойства (в том числе свойства объекта window) необходимо удостовериться, что ссылка на свойство начинается с имени объекта, как это показано в примере:

```
if (window.innerHeight) {    }
```

Еще один повод быть осторожным при проверке существования свойства — возможность того, что оно существует, но равно пустой строке или 0. Таким образом, надежнее проверять тип данных, сравнивая его с "undefined":

```
if (typeof myObj.myProperty != "undefined" ) {
    // У объекта myObj существует свойство myProperty
}
```

Если существует возможность того, что ни объект, ни свойство не существуют, нужно группировать условные операторы, чтобы проверять существование их обоих. Для этого сначала проверяется существование объекта, а затем — его свойства. Если объект не существует, проверка существования свойства пропускается:

```
if (myObj && typeof myObj.myProperty != "undefined" ) {
    // Существует объект myObj. и у него существует свойство myProperty
```

Если сначала проверять существование свойства, произойдет ошибка выполнения сценария.

В JavaScript имеется также специальная сокращенная запись, позволяющая избежать использования фигурных скобок в простых присваиваниях, применяющих условия. Синтаксис таков:

```
var myValue = (условие) ? значение1 : знэчение2:
```

Если условие истинно, выражение в правой части оператора присваивания будет равно первому значению. В противном случае — второму. Например:

```
var backColor = (temperature > 100) ? "red"    "blue":
```

В некоторых примерах интенсивно используется такая конструкция, иногда с несколькими уровнями вложения. Вот пример вложения:

```
var backColor - (temperature > 100) ? "red"    ((temperature < 80) ?
    "blue"    "yellow"):
```

Это выражение эквивалентно такому более длинному и удобочитаемому, но менее элегантному выражению:

```
var backColor :
if (temperature > 100) {
    backColor ≈ "red":
} else if (temperature < 80) {
    backColor = "blue",
} else {
    backColor = "yellow";
}
```

Если имеется множество вервей исполнения кода и отдельные условия также не являются простыми сравнениями, следует использовать конструкцию switch. В следующем примере рассматривается форма, в которой пользователь может выбирать размер продукта. После того как выбор сделан, обработчик события onchange вписывает соответствующую цену в текстовое поле:

```
function setPrice(form) {
    var sizeList = form.sizePicker;
    var chosenItem = sizeList.options[sizeList.selectedIndex].value;
    switch (chosenItem) {
        case "маленький" :
            form.price.value = "44.95":
            break;
        case "средний" :
            form.price.value = "54.95":
            break,
        case "большой" :
            form.price.value = "64.95":
            break.
        default:
             form.price.value = "Выберите размер"
    }
}
```

Если проверяемое выражение всегда равно одному из вариантов, ветку default можно опустить, но на время разработки страницы лучше оставить его для безопасности, выдавая сообщение о возможной ошибке сценария.

Смотрите также

Большая часть рецептов 15 главы используют сокращенную запись условных выражений для выравнивания несравнимых значений.

4.7. Обработка ошибок сценария

NN6 **IE5**

Задача

Необходимо незаметно для пользователя обработать ошибки выполнения сценария, тем самым предотвращая вывод сообщений об ошибках

Решение

Черновой, но обладающий обратной совместимостью способ предотвратить вывод сообщения об ошибке предполагает использование следующего кода в начале страницы:

```
function doNothing() {return true:}
window.onerror = doNothing;
```

Это не предотвратит ошибки компиляции сценария (то есть синтаксические ошибки, обнаруживаемые при загрузке страницы). Этот способ также не сможет показать программисту, где именно в коде прячется ошибка. Использовать такой код желательно только после того, как сценарий полностью отлажен. Для проверки работы сценария этот код необходимо удалить.

В IE 5, Netscape 6 и более поздних версиях этих браузеров имеется возможность более формальной обработки ошибок (исключений). Для того чтобы более поздние версии браузеров не споткнулись о специфический синтаксис, используемый для этих целей, такой код желательно встраивать в страницу с помощью тегов **<script>,** указывающих в качестве языка JavaScript **1.5 (language="JavaScript1.5").**

Операторы, которые могут привести к ошибке (выбросить исключение), оборачиваются в конструкцию **try/catch.** Исполняемые операторы лежат в секции try, а в секции **catch** находятся операторы, обрабатывающие возникающее исключение:

```
<script type="text/javascript" language="JavaScriptl.5">
function myFunc() {
    try {
        // операторы, которые могут выбросить исключение при некоторых
        // условиях
    }
    catch(e) {
        // Операторы, обрабатывающие исключение (объект, помещаемый
        //в переменную е
    }
}
</script>
```

Даже если в секции **catch** ничего не делать, возникающее в секции **try** исключение перестает быть фатальным. Выполнение функции будет продолжено, нужно только обеспечить независимость оставшегося кода от переменных в секции try. Можно аккуратно завершить выполнение функции, вызвав оператор return внутри блока **catch.**

Обсуждение

Каждое выброшенное исключение приводит к созданию нового объекта **Error.** Ссылка на этот объект передается в секцию **catch** через ее параметр. Операторы внутри секции **catch** могут проверять значения свойств этого объекта для определения дополнительной информации об ошибке. Только малая часть этих свойств внесены на текущий момент в стандарт ECMAScript, но некоторые браузеры поддерживают дополнительные свойства. Они содержат информацию того же рода, что можно видеть в сообщениях об ошибках. Таблица 4.2 содержит описание свойств объекта **Error** и информацию об их поддержке разными браузерами.

Таблица 4.2. Свойства объекта Error

Свойство	IE для Windows	IE для Mac	NN	Описание
description	5	5	—	Текстовое описание ошибки
fileName	—	—	6	URL файла, где произошла ошибка
lineNumber	—	—	6	Номер строчки кода с ошибкой
message	5.5	—	6	Текстовое описание ошибки (ECMA)
name	5.5		6	Тип ошибки (ECMA)
number	5	5	—	Собственный код ошибки Microsoft

Сообщения об ошибке не должны демонстрироваться пользователю. Свойства **description** или **message** нужно использовать для того, чтобы решить, как обрабатывать данное исключение. К сожалению, точное значение строки с сообщением не всегда одно и то же для одной ошибки. Например, при попытке обратиться к несуществующему объекту IE выдаст такое сообщение:

```
'myObject' is undefined
```

А Netscape 6 следующее:

```
myObject is undefined
```

Все это делает совместимую обработку исключений несколько сложной. Для того чтобы обойти эти проблемы, можно использовать проверку содержимого строки (например, с помощью регулярных выражений) на наличие фрагмента "defined" и имени объекта:

```
try {
    window.onmouseover = trackPosition;
}
catch(e) {
    var msg = (e.message) ? e.message : e.description;
    if (/trackPosition/.exec(msg) && /defined/.exec(msg)) {
        // Функция trackPosition не определена в этой странице
    }
```

Также можно умышленно выбрасывать исключения для использования обработки исключений в собственных сценариях. Показанная ниже функция — один

из вариантов функции проверки ввода числа в поле. Блок try проверяет правильность введенного значения. Если значение некорректно, создается новый объект Error, и с помощью оператора throw выбрасывается исключение. Естественно, выброшенное исключение немедленно перехватывается блоком catch, который выводит сообщение об ошибке и возвращает фокус ввода на поле:

```
function processNumber(inputField) {
    try {
        var inpVal = parseInt(inputField.value, 10);
        if (isNaN(inpVal)) {
            var msg = "Пожалуйста, введите число.";
            var err = new Error(msg);
            if (!err.message) {
                err.message = msg;
            }
            throw err;
        }Здесь можно безбоязненно работать с числом
    }
    catch (e) {
        alert(e.message);
        inputField.focus();
        inputField.select();
    }
}
```

Функции проверки такого рода могут вызываться как в обработчике события onchange, так и при проверке всей формы перед отсылкой, как это показано в главе 8.

4.8. Повышение производительности

NN2 IE3

Задача

Необходимо ускорить медлительный сценарий.

Решение

Заглатывая маленькие кусочки кода, интерпретатор JavaScript действует достаточно быстро, но при обработке тонн сложного кода большой степени вложенности может возникнуть заметная задержка, даже если все данные уже загружены.

Вот несколько полезных советов, призванных исключить узкие места в вычислительном процессе:

- избегайте использования функции eval();
- избегайте использования конструкции with;
- исключите повторное вычисление одних и тех же значений;
- для поиска значений в больших массивах используйте имитацию хэш-таблиц;
- избегайте многочисленных вызовов document.write();

Особенно необходимо заботиться о выполнении этих рекомендаций в циклах, где задержки умножаются.

Обсуждение

Одной из наименее неэффективных функций JavaScript является функция eval(). Эта функция преобразует строковое значение в ссылку на объект. Она оказывает помощь в тех случаях, когда по имеющейся строке, содержащей имя или идентификатор объекта, необходимо сформировать ссылку на объект. Например, если имеется последовательность изображений с именами menuImg1, menuImg2 и т. д., составляющих меню, может возникнуть желание создать функцию, восстанавливающую ссылку на объект:

```
for (var i = 0: i < 6: i++) {
    var imgObj = eval("document.menuImg" + i):
    imgObj.src = "images/menuImg" + i + "_normal.jpg";
}
```

К сожалению, использование функции eval() в этом цикле занимает много времени.

В тех случаях, когда необходимо ссылаться на элементы документа, почти всегда есть способ обойтись без использования eval(). В данном случае путь к решению дает коллекция (массив) изображений document.images. Вот исправленный, более обтекаемый код:

```
for (var i = 0: i < 6: i++) {
    var imgObj = document.images["menuImg" + i]:
    imgObj.src = "images/menuImg" + i + "_normal.jpg";
}
```

Если элемент документа имеет собственный идентификатор, можно получить ссылку на него, используя соответствующую коллекцию. Естественным выбором для браузеров, поддерживающих такой синтаксис, является применение функции document.getElementById(), имеющейся в стандарте W3C DOM. Но даже в более старых браузерах есть способ получить ссылку на объект по его имени, используя нужную коллекцию, например documentimages или form.elements (используя запись document.myForm.elements["имяЭлемента"]). Для пользовательских объектов смотрите приведенное ниже обсуждение хэш-таблиц. Выследив каждый вызов функции eval() в своем коде, вы получите быстрый и эффективный сценарий.

Другой враг производительности — конструкция with. Назначение этого управляющего оператора — указать, к какому объекту относятся операторы в блоке. Например, если имеется последовательность операторов, работающих непосредственно со свойствами и методами одного объекта, можно указать, что операторы внутри блока относятся к этому объекту. В следующем фрагменте сценария операторы в блоке вызывают метод объекта sort() и определяют значение свойства length:

```
with myArray {
    sortO:
    var howMany = length;
}
```

Да, это может показаться эффективным, но интерпретатор при выполнении каждого оператора выполняет дополнительные действия, поэтому не стоит использовать такую конструкцию.

Процессор загружает и вычисление каждого выражения или ссылки. Чем больше «точек» в ссылке на объект, тем больше времени уходит на разрешение ссылки. Таким образом, следует избегать повторяющихся длинных ссылок на объекты, особенно в цикле. Вот пример кода, демонстрирующего такое поведение:

```
function myFunction(elemID) {
    for (i = 0; i < document.getElementById elemID).childNodes.length; i++)
    [
        if (document.getElementById(elemID).childNodes[i].nodeType = 1) {
            // обработка элементов документа
        }
```

В процессе исполнения этой функции при каждом проходе цикла дважды вычисляется выражение document.getElementById(). Один раз это выражение вычисляется в самом начале цикла, при проверке условия выхода. Второй раз то же самое выражение вычисляется в операторе if внутри цикла. Кроме того, почти наверняка дополнительные операторы внутри цикла будут вычислять то же выражение, обращаясь к дочерним узлам элементов. Все это занимает драгоценные ресурсы процессора.

Ценой одной локальной переменной можно избавиться от повторных вычислений. Необходимо однократно вычислить повторяющуюся часть и впоследствии использовать вычисленное значение:

```
function myFunction(elemID) {.
    var elem = document.getElementById(elemID).
    for (i = 0; i < el em.childNodes.length; i++) {
        if (elem.childNodes[i].nodeType == 1) {
            // обработка элементов документа
```

Если же все вычисления внутри цикла ведутся только с дочерними узлами, можно еще больше ускорить выполнение цикла:

```
function myFunction(elemID) {
    var elemNodes = document.getElementById(elemID).childNodes;
    for (i = 0; i < elemNodes.length; i++) {
        if (elemNodes[i].nodeType == 1) {
            // обработка элементов документа
        }
    }
}
```

Дополнительное преимущество данного подхода — уменьшение размера кода. Таким образом, если вы видите в коде сценария повторяющиеся выражения, стоит рассмотреть возможность их однократного вычисления.

Следующим шагом в повышении производительности является исключение таких поглощающих процессорное время операций, как перебор элементов массивов, особенно многомерных и массивов из объектов. Если массив достаточно велик (скажем, более чем из 100 элементов), даже средняя задержка при поиске может быть весьма заметна. Вместо этого можно использовать методику, продемонстрированную в рецепте 3.9, когда для массива однократно составляется имитация хэш-таблицы. Процесс построения такой таблицы можно совместить с загрузкой страницы, при этом задержка на построение таблицы будет незаметна на фоне всего времени загрузки страницы. Зато потом поиск в массиве будет происходить практически мгновенно, даже если в массиве имеются сотни записей.

Последний совет касается метода document.write(), формирующего содержимое страницы при загрузке. Этот метод следует рассматривать как относительно медленный метод ввода-вывода, поэтому следует избегать его частого использования. Если необходимо вывести на страницу большое количество HTML-кода, накопите текст в строковой переменной и выведите его одним вызовом document.write().

Смотрите также

В рецепте 3.9 описываются детали использования имитированных хэш-таблиц для массивов. Рецепт 3.13 демонстрирует тот редкий случай, когда использования функции eval() не избежать.

5 Определение возможностей браузера

5.0. Вступление

Пожалуй, одной из самых сложных проблем, стоящих перед каждым автором, нуждающимся в сценариях или возможностях DHTML, является выбор тех возможностей, которые обеспечат работу сценария на наибольшем наборе браузеров. Не так давно, определяя версию браузера, авторы заботились прежде всего о поддержке таких «новых» элементов HTML, как таблицы или фреймы. Сейчас ситуация заметно усложнилась. Несмотря на провозглашаемую совместимость с промышленными стандартами, каждый браузер реализует некоторые возможности по-своему, предлагая свое видение стандартов. Иногда производители браузеров предлагают собственные, несовместимые со стандартами возможности.

Есть две области, в которых изменения от версии к версии наиболее сильны. Это ядро языка JavaScript (описанное в главах с 1 по 4) и объектная модель документа (это элементы страницы, которыми можно управлять, используя JavaScript). То, что сейчас можно увидеть на многих страницах, например рисунки, меняющие цвет или оттенок при наведении мыши, не было возможно в ранних версиях браузеров. Изображения не рассматривались как объекты DOM, что не позволяло сценариям менять значения свойства, содержащего путь к различным файлам *.jpg или *.gif. Даже возможности, скрытые от пользователя, такие как типы данных языка сценариев, изменялись от версии к версии. Элементы языка, которые, как может показаться опытному программисту, присутствовали в языке с самого начала, существовали не всегда. Примером могут служить массивы и оператор switch. На сегодняшний день основная опасность кроется в том, что сценарий, использующий DHTML в контексте последних версий современных браузеров, может не работать на более старых версиях.

Это осложняется тем, что, несмотря на доступность новых версий браузеров, пользователи не всегда стремятся переходить на них. Стоимость обновления браузеров может оказаться слишком большой. Многие пользователи применяют стабильные браузеры, надежность которых подтверждена годами применения. Такие браузеры не обновляются каждые 12-18 месяцев. Автор может легко

впасть в эйфорию при появлении нового браузера, забыв о том, что масса пользователей, применяющих новую версию браузера, достигнет критической примерно через полгода — год после выхода.

Эти и другие факторы приводят к разнородности имеющихся по всему миру браузеров. Задача автора веб-приложений — балансировать между желанием украсить содержимое с использованием различных возможностей DHTML и необходимостью обеспечивать нормальное отображение содержимого на менее современных браузерах. Один из шагов к достижению равновесия — определить, насколько широкий набор систем должен поддерживать продукт. В идеале, конечно, можно сделать множество страниц, каждая из которых будет оптимизирована под конкретную комбинацию производителя, версии браузера и операционной системы. Но даже имея сложные средства управления содержимым, простое изменение текста на странице может превратиться в невообразимый кошмар. В более жизненной ситуации HTML-страница (или серверная процедура под управлением PHP, ASP, JSP или чего-либо еще) сама может подстраиваться под индивидуальные особенности браузера и операционной системы. Это и называется определением возможностей браузера.

К сожалению, нет простых и ясных правил, которым можно было бы следовать при учете различий браузеров. Каждое сочетание содержимого и целевой аудитории формирует отдельные требования к совместимости. Полезно иметь это в виду, прежде чем начать размышлять о том, насколько следует различать браузеры в проекте. Фактически это первая вещь, о которой необходимо задуматься, начиная новый проект.

Выработка стратегии

Первым этапом разработки должно стать создание макета внешнего вида и содержимого страниц. Затем нужно четко представить себе идеальное взаимодействие с пользователем, придающее смысл содержанию. Например: будет ли навигация по сайту с помощью всплывающих подменю, связанных с основным меню, более быстрой? Нужно ли предлагать возможность пользовательской настройки вида страницы? Будет ли пользователям удобно и интересно строить свое сетевое окружение из компонентов, перетаскивая мышью значки, обозначающие модули?

После того как такой список пожеланий получен, настало время определить, какие версии браузеров поддерживают необходимые возможности, а затем учесть, какие из этих возможностей будут доступны пользователям. Возможно, вам захочется добавить на страницу все самые разнообразные дополнения к пользовательскому интерфейсу, доступные в новейших браузерах, а пользователям более старых — предложить более простое представление. Многие из сценариев DHTML можно включать в страницы так, чтобы один и тот же HTML-файл работал как с новыми, так и с более старыми браузерами, предоставляя пользователям первых дополнительные возможности.

Для того чтобы воплотить в жизнь эти грандиозные замыслы, необходимо включить в документ сценарии для определения возможностей браузера. Настройка может представлять собой просто формирование различных путей загрузки

сценария для Navigator и Internet Explorer. В зависимости от типа и версии браузера можно выбрать разные пути к документам. Используя этот метод, можно также учитывать, поддерживается ли вообще в браузере возможность выполнения сценариев. Тогда начальная страница может играть роль фильтра для различных типов браузеров.

Когда нет JavaScript

Для новичка в создании сценариев, работающих на машине пользователя, не столь необычен вопрос о том, как сценарий может определить отсутствие возможности исполнять JavaScript. Только после некоторого размышления становится понятна нелепость попытки использовать JavaScript для принятия решения о том, не игнорирует ли браузер все сценарии вообще.

JavaScript может быть недоступен у пользователя по двум причинам: по выбору или по принуждению. Наиболее распространен случай, когда браузер ничего не знает о JavaScript. Это касается не только текстовых браузеров наподобие Lynx или других браузеров специального назначения. Сейчас все возрастает число портативных беспроводных устройств, на которых используются браузеры с плохой поддержкой сценариев (по сравнению с полноценными браузерами) или вообще без таковой.

Несмотря на новейшие «примочки» последних браузеров, а также наличие «рекомендованных» путей создания нужных конструкций, многие тысячи страниц в Сети используют методики, давно канувшие в Лету. Производители браузеров вынуждены обеспечивать обратную совместимость (за редким исключением) своих продуктов с более ранними технологиями. К сожалению, из-за поддержки устаревших технологий отдельные авторы думают, что их использование в порядке вещей, тем самым у них отсутствует стимул изучать новые. С другой стороны, профессиональный разработчик, слепо полагающийся на стандарты, может оказаться еще более беспечным, поскольку производители браузеров до сих пор не привели свои продукты к стандарту или же реализовали их весьма своеобразно.

Второй причиной отсутствия поддержки JavaScript может быть то, что пользователь запретил исполнение сценариев по личным мотивам или по соображениям безопасности. Когда исполнение сценариев отключено в браузере, который в противном случае смог бы их выполнять, любой HTML-код, ограниченный парой тегов <noscript> </noscript>, отображается на странице. Как раз сюда и следует поместить сообщение, объясняющее пользователям, что они теряют:

```
<noscript>
<p>
Если вы разрешите выполнение JavaScript, то сможете насладиться
удобными ярлыками, ускоряющими доступ. Чтобы узнать, как включить JavaScript
в своем браузере, щелкните на одной из ссылок:
<a href="jsiewin.html">Internet Explorer для Windows</a>,
<a href="jsiemac.html">Internet Explorer для Macintosh</a>,
<a href="jsnetscape.html">Netscape Navigator</a>.
</p>
</noscript>
```

Пользователи браузеров без интерпретатора сценариев увидят то же самое сообщение. Используя несложную серверную программу, можно оказать посетителям еще большую помощь. Анализируя значение параметра HTTP USERAGENT, содержащегося в запросе на страницу, сервер может определить тип браузера и операционной системы, благодаря чему можно будет поместить в секцию **<noscript>** только одну ссылку на инструкции по включению JavaScript Это особенно полезно для тех пользователей, которые не знают, каким браузером они пользуются. Так несложная серверная программа может облегчить жизнь неопытным пользователям.

Скрытие сценариев от не поддерживающих их браузеров

Хотя неподдерживающие сценарии браузеры игнорируют теги **<script>** и **</script>**, они зачастую не умеют игнорировать код, помещенный между этими тегами. Не получая дополнительных указаний, такие браузеры зачастую отображают код сценария, как если бы это был текст страницы. Чтобы предотвратить подобные действия, необходимо заключить код в теги комментариев:

```
<script language="JavaScript">
<!-
Здесь расположен код сценария
//->
</script>
```

Открывается HTML-комментарий как обычно, символами <!—. Но тег конца комментария (—>) приводит к синтаксической ошибке в сценарии, поэтому его следует предварять символами комментария JavaScript (//). Если страница рассчитывается на просмотр в браузерах, неподдерживаюших сценарии, каждая пара тегов **<script>** должна включать теги комментария HTML. Тем не менее нужно быть готовым к тому, что некоторые особенно устаревшие браузеры, используемые постоянно уменьшающимся числом пользователей, могут не понять эти вполне корректные символы HTML и отобразят содержимое сценария прямо на странице, вне зависимости от действий программиста.

Описанная здесь техника маскировки кода предотвращает только отображение кода на странице в некоторых браузерах. Этот метод не может помешать любопытному пользователю увидеть содержимое страницы или сценария. И предупреждая возможный вопрос, следует упомянуть о том, что не существует надежного способа помешать пользователям просматривать код сценария. Хотя и существует множество способов запутывания кода, настойчивый посетитель тем или иным путем всегда сможет просмотреть сценарий.

Определение версии JavaScript

Существует несколько неточный способ определения версии браузера, основанный на возможности выбирать тот или иной путь вычислений в зависимости от версии языка JavaScript, встроенного в браузер. Этот способ называется определением браузера. Он применялся на ранних порах развития JavaScript, когда

перемены от версии к версии в языке сценариев были тесно связаны с версиями двух основных браузеров: Internet Explorer и Netscape Navigator. Например, версия JavaScript 1.2 использовалась в Navigator 4 и Internet Explorer 4.

С тех пор взаимосвязь между версией JavaScript и возможностями браузеров, определяемыми различиями в модели документа, стала слишком слабой, чтобы ее можно было использовать. Сейчас не рекомендуется различать версию языка, но поскольку на многих страницах используется эта методика, вниманию читателя предлагается небольшое описание того, как это работает.

Как и код HTML, код сценария загружается в браузер в том порядке, в котором он расположен на странице (то есть сверху вниз). Если браузер встречает более одной функции с одним и тем же именем, та, что описана позднее, заменяет более раннюю. Но браузер интерпретирует теги <script> только для тех версий JavaScript, о которых он знает. В качестве примера рассмотрим следующий набор независимых тегов:

```
<script language="JavaScript" type="text/javascript">
function myFunction() {
    код функции
}
</script>
<script language="JavaScript1.2" type="text/javascript">
function myFunction() {
    код функции
}
</script>
```

Браузер, имеющий представление только о JavaScript версий 1.0 и 1.1, сохранит в памяти первое определении функции myFunctionO, которую затем можно будет вызывать в обработчике события или из основного кода сценария. Браузер, понимающий JavaScript 1.2 (или более поздние версии), сначала загрузит первое определение функции, но затем тут же заменит его новым определением. Поэтому при вызове где-либо в коде функции myFunction() будет выполнено только втрое определение.

Проверка объектов — правильный путь

Использование версии и названия браузера, информации об операционной системе требует от программиста знаний о том, как реализована поддержка отдельных объектов, свойств и методов в каждой из ситуаций. Учитывая количество браузеров, бороздящих Интернет, становится ясно, что невозможно знать, какая из современных или будущих версий каждого из них поддерживает необходимые возможности объектной модели. Здесь необходима методика, называемая проверкой объектов. Как показано в рецептах 5.6 и 5.7, сценарий может удостовериться в том, что необходимая возможность поддерживается, прежде чем использовать ее.

Проверку объектов следует использовать осторожно. Не думайте, что поддержка одной возможности какого-либо браузера влечет за собой поддержку всех остальных его возможностей. К примеру, существует порочная практика полагать, что браузер, поддерживающий коллекцию document.all, является Internet

Explorer версии 4 или выше. Те, кто знакомы с этим классом браузеров, работающих в среде Windows, обычно имеют опыт использования широкого диапазона возможностей IE DOM. Однако некоторые из этих возможностей недоступны как в версиях IE, работающих на операционных системах, отличных от Windows, так и в браузере Opera, который, как и IE, поддерживает свойство document.all. С другой стороны, не нужно быть гуру, чтобы понять, что поддержка свойства document.all автоматически влечет за собои поддержку document.all.length, так как все коллекции имеют свойство length.

Разумное использование методики проверки объектов, таким образом, требует хорошего набора признаков совместимости с языком и объектной моделью.

Задание глобальных переменных

То, насколько часто сценарий вынужден выполнять проверки версии браузера или объектов, зависит от его величины и от того, насколько часто нужно выполнять действия, различающиеся для разных браузеров. В большинстве случаев проверка браузера происходит через объект navigator, иногда в нее вовлекаются составные проверки (например, когда проверяется определенный тип браузера и его версия). Поскольку в большинстве случаев при учете версии браузера используется ветвление с помощью оператора if, гораздо разумнее задать несколько глобальных переменных в начале сценария и использовать затем их значения как флаги. Пусть в операторах присваивания производятся нужные проверки, а глобальным переменным присваиваются булевские значения true или false. После этого флаги можно использовать в операторах ветвления (if):

```
var isMacOS = (navigator.userAgent.indexOf("Mac") != -1);

if (isMacOS) {
    Код, предназначенный специально для Macintosh
} else {
    Другой код
}
```

То же можно сделать и при проверке объектов, тем самым уменьшая количество громоздких вычислений:

```
var isW3C = (document.getElementById) ? true : false;
```

Приобретя некоторый опыт, вы научитесь понимать, какие из проверок нужны для каждой конкретной задачи. И тогда формирование таких глобальных переменных в начале каждого сценария станет вашей второй натурой.

DHTML и доступность

Во многих странах действуют законы, предписывающие работодателям и организациям создавать необходимые условия для служащих, клиентов и посетителей с ограниченными возможностями. Такие пользователи нередко посещают страницы с помощью браузеров, имеющих только клавиатурное управление или же отображающих содержимое страницы с помощью синтезатора речи. В этом случае приложение, использующее для подачи содержимого технологию DHTML,

может нарушить такие законы. Тесно связана с этой проблемой проблема доступности содержимого для поисковых машин и прочих автоматических средств, бороздящих Сеть в поисках информации для своих баз данных. Как правило, они следуют обычным HTML-ссылкам, не исполняя сценариев. Таким образом, им доступен только HTML в том виде, в котором он поступает с сервера. Поэтому, если все переходы между страницами реализованы с помощью сценариев, поисковая машина так и не сможет изучить сайт.

5.1. Определение производителя браузера

NN2 **IE3**

Задача

Необходимо, чтобы для пользователей Netscape выполнялась одна ветвь сценария, а для пользователей Internet Explorer — другая.

Решение

Для того чтобы определить производителя браузера, можно использовать свойство navigator.appname. Следующие два оператора устанавливают необходимые глобальные переменные:

```
var isNav = (navigator.appName == "Netscape");
var isIE = (navigator.appName =="Microsoft Internet Explorer");
```

Обсуждение

Свойство navigator.appname содержит строку, определяющую производителя браузера. Netscape Navigator и Mozilla всегда содержат в этом свойстве значение "Netscape", а IE — "Microsoft Internet Explorer". Эти строки всегда имеют такой вид, поэтому соответствующие им операторы сравнения чувствительны к регистру.

Выйдя за пределы сфер NN и IE, можно встретить некоторые странные искажения, возникающие в этой схеме. Например, браузер Opera в зависимости от настройки может содержать в свойстве appName значения: "Netscape", "Microsoft Internet Explorer" или "Opera". При настройке как IE или Netscape Opera начинает интерпретировать сценарии как соответствующий браузер (по крайней мере, в простейших случаях). Для того чтобы с уверенностью определить, является ли браузером Opera, необходимо покопаться в свойстве navigator.userAgent, содержащем больше сведений о браузере. Вот значение этого свойства, принятое по умолчанию в Opera 6:

```
Mozilla/4.0 (compatible: MSIE 5.0: Windows 98) Opera 6 03 [en]
```

Как видно, информация о том, что по-настоящему браузером является Opera, запрятана в этой строке. Вместо того чтобы проверять значение определенного

участка строки (которое, к тому же, со временем может измениться), надежнее проверить, содержится ли подстрока "Opera" в свойстве navigator.userAgent:

```
var isOpera = (navigator.userAgent indexOf("Opera") != -1);
```

Что касается Internet Explorer, следует учитывать наличие существенных различий между версиями для Windows и Macintosh. Поэтому также может потребоваться определять операционную систему (это описывается в рецепте 5.5).

Смотрите также

Рецепты с 5.2 по 5.4 более подробно описывают проверки версии браузера. В рецепте 5.5 демонстрируется определение операционной системы.

5.2. Определение ранних версий браузеров

NN2 **IE3**

Задача

Необходимо выполнять отдельную ветвь кода для ранних версий браузеров.

Решение

Начиная с четвертой версии, основные браузеры в свойстве navigator.appVersion содержат полную информацию о версии браузера, а также некоторые сведения об операционной системе. Для того чтобы определить основную версию браузера, можно использовать функцию parseInt(), возвращающую в данном случае номер:

```
var isVer4 = (parseInt(navigator.appVersion) ==4);
```

Чтобы узнать номер версии, включая дополнительный номер (цифры справа после точки, например 4.74), можно использовать функцию parseFloat():

```
var isVer4_74 = (parseFloat(navigator.appVersion) == 4.74).
```

Если необходимо, чтобы переменная означала наличие некоторой минимальной версии, вместо оператора проверки равенства (==) следует использовать оператор >=:

```
var isVerMin4 = (parseInt(navigator.appVersion) >= 4).
var isVer4_5Min = (parseFloat(navigator.appVersion) >= 4.5),
```

Обсуждение

Основные браузеры, вплоть до версий Microsoft Internet Explorer 4.0 и последних из семейства Netscape Navigator 4 (на момент написания это была версия 4.79), содержат в строке navigator.appVersion информацию, начинающуюся с точных сведений о версии браузера. Например:

```
Microsoft Internet Explorer 4.01. работающий под управлением Windows 98
    4.0 (compatible: MSIE 4.01: Windows 98)
Netscape Navigator 4.79 для Macintosh PowerPC
    4.79 (Macintosh: U: PPC)
```

Однако для последующих версий первые символы свойства navigator.appVersion уже не соотносятся с реальным номером версии браузера. Вместо этого числа в начале строки отражают теперь поколение программного ядра браузера. Так, даже у Internet Explorer 6.0 для Windows строка navigator.appVersion начинается с чисел 4.0.

Новейшие версии браузеров Netscape построены на основе полностью нового ядра (ядро Gecko, разработанное Mozilla). По ряду исторических причин, это поколение кода считается пятым. Поэтому в браузерах Netscape 6 и 7, использующих это ядро, а также в Mozilla строка navigator.appVersion начинается с 5.0 и не соответствует реальной версии браузера.

Для того чтобы точно определить версию нового браузера, необходимо использовать другие методы, которые, к сожалению, не работают на всех браузерах. Эти методы описаны в рецептах 5.3 и 5.4.

Смотрите также

Рецепты 5.1, 5.3, 5.4 содержат дополнительные советы по определению версии браузера.

5.3. Определение версии Internet Explorer

NN нет IE 3

Задача

Необходимо определить наличие минимальной версии Internet Explorer.

Решение

Полный номер версии можно получить, разобрав строку, хранящуюся в свойстве navigator. userAgent. Internet Explorer идентифицирует себя символами "MSIE", за которыми следуют: пробел, полный номер версии и точка с запятой. Вот функция, выделяющая числовую часть нужной информации:

```
function getIEVersionNumber() {
    var ua = navigator.userAgent;
    var MSIEOffset = ua.indexOf("MSIE ");
    if (MSIEOffset — -1) {
        return 0:
    } else {
        return parseFloat(ua.substring(MSIEOffset + 5.
            ua.indexOf(":", MSIEOffset)));
    }
}
```

Возвращаемое этой функцией значение можно использовать для того, чтобы установить глобальный флаг. Этот флаг может применяться где-то в коде сценария, в операторах ветвления. Например, так можно будет выполнить код только для браузеров, совместимых с IE 5 или выше:

```
var isIE5Min - getIEVersionNumber() >= 5: // Флаг установлен

if (isIE5Min) {
    // Операторы. которые будут выполнены только в IE 5 или выше
}
```

Обсуждение

Поскольку приведенная в решении функция возвращает числовой результат, она не может быть использована для тех случаев, когда версия включает в себя букву. Например, в бета-версиях за номером версии следует буква B. Если не вызывать функцию parseFloat(), можно получить строковую информацию о версии, включающую такие детали. Обычно требуется числовое значение, которое можно использовать для сравнения, как это делалось выше.

Использование числового сравнения при анализе номера версии — это палка о двух концах. С одной стороны, необходимо не забывать применять для контроля версии оператор «больше или равно» (>=) вместо оператора проверки на равенство (==). Использование оператора проверки на равенство — частая ошибка у тех, кто пытается создать код, который бы работал только на последних браузерах. Проблема в том, что в еще более новой версии, когда она появится, сравнение даст отрицательный результат и код выполнен не будет. С другой стороны, применение оператора >= предполагает, что позднейшие версии продолжат поддержку нужных возможностей. Несмотря на то что браузеры обычно обладают обратной совместимостью, те, кто знает об исчезновении объекта layer между 4-й и 6-й версиями Netscape Navigator, понимают, что такое предположение может привести к проблемам. Это еще один повод выполнять в большинстве случаев не проверку версии, а проверку объектов. Тем не менее проверка версий полезна в тех случаях, когда в сценарии необходимо обойти ошибку в определенной версии.

Смотрите также

Рецепты 5.2 и 5.4 предлагают дополнительные советы по определению версии браузера. Рецепт 2.1 посвящен использованию функции parseFloat(). В рецепте 1.5 применяется метод indexOf().

5.4. Определение версии Netscape Navigator

NN2 **IE нет**

Задача

Необходимо выполнять разные ветви кода для разных версий Netscape Navigator.

Решение

Для версий, предшествующих Netscape 6, наиболее доступная информация о версии хранится в начале строки navigator.appVersion. Тем не менее, начиная с Netscape 6, информация о версии хранится в новом свойстве объекта navigator, vendorSub. Ниже показан пример функции, определяющей номер версии для любого поколения:

```
function getNNVersionNumber() {
    if (navigator.appName -= "Netscape") {
        var appVer = parseFloat(navigator.appVersion);
        if (appVer < 5) {
            return appVer:
        } else {
            if (typeof navigator.vendorSub !- "undefined") {
                return parseFloat(navigator.vendorSub);
            }
        }
    }
    return 0:
}
```

Эту функцию можно использовать, чтобы установить глобальный флаг, применяемый в операторах ветвления кода. Вот пример того, как код будет выполнен только в том случае, если браузер заявляет совместимость с Netscape 6 или выше:

```
var isNN6Min = getNNVersionNumber() >= 6: // Флаг установлен

if (isNN6Min) {
    // Операторы, выполняемые только в Netscape 6 или выше
```

Обсуждение

Со временем все возрастает угроза того, что проверка на определенную версию и название браузера может помешать выполнению кода в браузерах, которые в действительности используют подходящее программное ядро. Все основные производители браузеров, включая Microsoft, Netscape (Mozilla) и Opera, разрабатывают свой код таким образом, чтобы иметь возможность встраивать функциональность браузера в другие типы устройств. Например, ядро Gecko, используемое в Navigator 6 и выше, может применяться в карманных компьютерах или других устройствах. Такой браузер, скорее всего, не будет использовать имя Netscape или ту же систему версий. Конечно, отдельное ядро включает в себя и другие свойства объекта navigator, product и productSub, позволяющие определять ядро Gecko и его версию. Это еще одна причина того, что проверка версии и браузера уже не столь привлекательный способ, как когда-то, если только не использовать такие проверки для обхода известных ошибок в существующих версиях.

Смотрите также

Рецепты 5.2 и 5.3 содержат дополнительные советы по проверке браузера Рецепт 2.1 демонстрирует использование функции parseFloat()

5.5. Определение ОС клиента

NN2 IE3

Задача

Необходимо использовать стили или другие особенности содержимого, предназначенные для отдельной операционной системы.

Решение

Значение, содержащееся в свойстве navigator.userAgent, обычно включает в себя некоторую информацию об операционной системе, под управлением которой работает браузер. К сожалению, не существует стандарта на названия операционных систем, которые можно было бы искать в этом свойстве. Названия различаются не только между браузерами разных производителей, они со временем меняются и в различных версиях одного браузера.

Самый широкий разброс в представлении платформы Windows. Например, Windows 98 может записываться как «Win98» или «Windows 98». Наилучшим решением является проверка значения свойства на наличие в нем строк, постоянных для некоторой операционной системы. К примеру, все версии браузеров для Macintosh содержат в свойстве navigator.userAgent строчку "Mac". Аналогично, все версии для Windows содержат в этом свойстве фрагмент "Win", но это верно для всех версий Windows, начиная с 3.1 и кончая XP. Более того, все версии Navigator для Unix содержат в нем строку "X11". Таким образом, если необходима приблизительная оценка, можно использовать следующие операторы для установки нужных логических флагов:

```
var isWin = (navigator.userAgent.indexOf("Win") != -1);
var isMac = (navigator.userAgent.indexOf("Mac") != -1);
var isUnix = (navigator.userAgent indexOf("X11") != -1);
```

Что касается Linux, то браузеры под этой операционной системой содержат в свойстве navigator.userAgent как строку "X11", так и "Linux".

Обсуждение

Как можно видеть, определение операционной системы — довольно сложное дело, поэтому должно использоваться с осторожностью. Несмотря на то что браузеры некоторых производителей содержат в свойстве navigator.userAgent более подробную информацию об операционной системе (например, номер выпуска ОС),

нельзя предполагать, что эта же информация имеется и в других браузерах для той же ОС. К примеру, Windows XP идентифицирует себя как одна из версий Windows NT. На Macintosh Netscape 6 и более поздние версии сообщают значительное количество информации, различая Mac OS Ч и более ранние версии. В то же время IE не различает версий Mac OS.

Navigator 6 и более поздние версии имеют еще одно дополнительное свойство объекта navigator, oscpu, которое содержит часть значения user Agent, относящуюся к операционной системе и центральному процессору. При работе под управлением Windows это свойство содержит только информацию об операционной системе, например, "Win98" или "Windows NT 5.1". В случае Mac это свойство всегда включает сведения о классе центрального процессора ("PPC") и названии операционной системы ("Mac OS X").

Один раз установив глобальные переменные-флаги так, чтобы они содержали необходимую вам информацию, можно использовать их для установки настроек, зависящих от операционной системы, таких как размеры шрифтов. Например, дизайнерам хорошо известен тот факт, что шрифт одной и той же заданной в пунктах высоты на дисплее Macintosh выглядит более мелким, чем в Windows. Таким образом, если вы рассчитываете задавать размеры шрифтов на странице в пунктах, можно добиться того, чтобы она выглядела одинаково на разных системах, динамически формируя таблицу стилей, используя для установки размера шрифта значения флагов:

```
<script language="JavaScript" type="text/javascript">
document.write("<style type='text/css'>");
document.write("body {font-size:" + ((isMac) ? "12" : "10") + "pt}");
document.write("</style>");
</script>
```

Аналогичные конструкции можно использовать для позиционирования элементов в зависимости от операционной системы, если это необходимо. На практике можно взять вложенные условные операторы для получения трехвариантного переключения. Например, чтобы установить значение переменой, содержащей смещение между элементами в пикселях, различным для трех основных операционных систем, можно применить следующий код:

```
var elemOffset = (isMac) ? 15 : ((isWin) ? 12 : 10);
```

После его выполнения значение переменной elemOffset будет равно 15 для Mac, 12 для всех вариантов Windows и 10 для всех остальных ОС (в основном это Unix).

Смотрите также

Рецепт 5.1 под одну определенную операционную систему. Особенно подходит в тех случаях, если у вас возникают проблемы из-за одной определенной версии браузера. Рецепт 11.5 предлагает улучшенные таблицы стилей для отдельных ОС. В рецепте 4.6 рассказывается о ветвлении выполнения кода.

5.6. Проверка поддержки объектов

NN2 **IE3**

Задача

Необходимо, чтобы сценарий работал на всех браузерах, поддерживающих требующиеся ему объекты, а на остальных корректно уменьшал свою функциональность.

Решение

Для этого необходимо окружить операторы, использующие потенциально неподдерживаемые объекты, операторами if, проверяющими существование этих объектов. Проверяемыми объектами могут быть как основные объекты ядра языка JavaScript, так и объекты DOM. Проведение проверок такого рода упрощает то, что ссылка на несуществующий объект эквивалентна логическому значению false в условном операторе. Очень часто проверку объектов применяли в ранних версиях браузеров в сценариях, работающих с элементам img для формирования изменяющихся при наведении указателя мыши изображений (это описывается в главе 12). Поддержка объекта, соответствующего элементу img, была очень неровной вплоть до четвертой версии браузеров. Проверка же всех возможных версий всех браузеров как минимум утомительна. Вместо того чтобы использовать такую проверку, все обращающиеся к объекту img операторы сценария можно обернуть конструкцией if, проверяющей существование массива document/images:

```
function rollover(imgName, imgSrc) {
    if (document.images) {
        document.images[imgName].src = imgSrc;
    }
}
```

Эту функцию можно использовать в обработчике события onmouseover для гиперссылки, обрамляющей нужное изображение:

```
<a href="product.html"
 onmouseover="rollover('products', 'images/products_on.gif'); return false"
 onmouseout="rollover('products', 'images/products_off.gif'); return false">
<img src="images/products_off.gif" border=0 alt="Products Page"></a>
```

Такая проверка работает корректно благодаря тому, что объект document в браузерах, в которых объект img доступен для обработки в сценариях, обязательно имеет связанный с ним массив images[]. Если на странице нет изображений, массив пуст, но все равно существует, хотя его размер и равен нулю. Если же массив вообще не существует, условие в операторе if не выполняется и операторы функции не производят никаких действий. Браузеры, которые могли бы «подавиться» ссылкой на несуществующий объект img, теперь аккуратно обходят опасные операторы.

Обсуждение

Проверка объектов освобождает программиста от необходимости тщательного исследования версии браузера. Но эту технику необходимо применять так, чтобы не поддерживающие нужные объекты браузеры корректно уменьшали свою функциональность. Это значит, что следует предвидеть ситуацию, когда сценарий запускается на браузере, поддерживающем только часть нужных объектов, а не все. В качестве примера рассмотрим сценарий, в котором производятся какие-то сложные вычисления. При этом процесс вычислений периодически вызывает внешние функции, получая от них вычисленное значение. Если одна из этих внешних функций выполняет проверку объектов, то как отреагирует основной поток вычислений на невозможность получить от функции корректное значение? Две показанные ниже функции написаны без использования преимуществ, которые дает проверка объектов. Здесь главная функция сначала вызывает подпрограмму, вычисляющую площадь всех изображений на странице, после чего выводит полученное значение в текстовое поле на форме:

```
function getImgAreas() {
    var result:
    // Устанавливаем начальное значение переменной.
    // чтобы впоследствии ее можно было использовать для накопления
    result = 0:
    // Перебор всех объектов img на странице
    for (var i - 0; i < document.images.length; i++) {
        // Накопление суммы площадей
        result +- (document.images[i].width *
           document.images[i].height):
    }
    return result:
}
function reportImageArea() {
    document.reportForm.imgData.value = output:
}
```

Браузер, не имеющий представления об объекте элемента img, при вызове функции getlmgAreasO сообщит об ошибке. Даже если сообщение об ошибке будет спрятано от пользователя (см. рецепт 4.7), пользователь все равно будет ожидать появления результата в текстовом поле, но ничего не произойдет.

Более опытный программист учтет, что вышеприведенный сценарий может привести к ошибке выполнения на очень старых браузерах, в которых ошибки сценария довольно навязчивы. Чтобы предотвратить такие сбои, автор должен сделать две модификации. Первым делом нужно поместить потенциально опасные операторы функции getlmgAreasO в блок, проверяющий наличие объекта. Во-вторых, главная функция должна теперь учитывать возможность того, что getlmgAreasO непроизведет никаких вычислений:

```
function getlmgAreasO {
    var result:
```

```
    // Проверка того, что браузер поддерживает объекты img
    if (document.images) {
        // Устанавливаем начальное значение переменной,
        // чтобы впоследствии ее можно было использовать для накопления
        result = 0;
        // Перебор всех объектов img на странице
        for (var i = 0; i < document.images.length; i++) {
            // Накопление суммы площадей
            result += (document.images[i].width *
                document.images[i].height);
        }
    }
    // Возвращаемое значение является либо числом, либо null
    return result;
}
function reportImageArea() {
    // Сохраняем результат для дальнейшего анализа
    var imgArea = getImgAreas();
    var output;
    if (imgArea ==null) {
        // Сообщение, формируемое для браузеров, не поддерживающих
        // объект img
        output = "неизвестно";
    } else {
        output = imgArea;
    }
    document.reportForm.imgData.value = output;
}
```

Обратите внимание на то, что главная функция не делает никаких проверок перед тем, как работать с полями формы. Опытный разработчик сценариев (или тот, у кого есть хороший справочный ресурс по DOM) должен знать, что синтаксис, применяемый для доступа к текстовым полям формы, обладает обратной совместимостью вплоть до самых ранних версий.

Тем не менее проверка объектов выручает не всегда. Иногда необходимо использовать проверку наличия нужного свойства или метода. Например, в приведенной подпрограмме не предполагалось никаких проблем при обращении к свойствам элемента img: width и height. Причина в том, что эти свойства были доступны у элемента img с самых ранних версий DOM. Но некоторые другие свойства поддерживаются не всеми браузерами, поддерживающими объект **img**, например свойство **alt**. Если сценарий должен обращаться к свойству **alt**, необходимо производить дополнительные проверки свойств, как это показано в рецепте 5.7.

Смотрите также

Описание проверки наличия нужного объекта или метода смотрите в рецепте 5.7. Рецепт 4.6 описывает специальные значения, которые можно использовать в логических операторах вместо **true** или **false**.

5.7. Проверка наличия свойства или метода

NN2 IE3

Задача

Необходимо, чтобы сценарий выполнялся на всех браузерах, поддерживающих нужные свойства и/или методы. На других браузерах сценарий должен корректно уменьшать свою функциональность.

Решение

Следует окружить операторы, использующие потенциально неподдерживаемые свойства или методы условными операторами, проверяющими возможность выполнения кода. Свойства и методы, существование которых проверяется, могут принадлежать как объектам ядра JavaScript, так и объектам DOM. Такая проверка немногим более сложна, чем проверка существования объекта самого по себе. Если при проверке наличия свойства или метода не существует сам объект, у которого они проверяются, происходит ошибка выполнения сценария. Поэтому необходимо предварительно проверить существование объекта (если этот объект не обладает полной обратной совместимостью):

```
if (objectTest && propertyTest) {
    // Здесь можно работать со свойством
}
```

Такая комбинация работает правильно, поскольку, получив в первом выражении значение false, сценарий пропустит выполнение второй проверки.

Кроме того, не рекомендуется использовать простую форму проверки существования свойства объекта в условном операторе if. Причина этого в том, что некоторые вполне законные свойства могут содержать нулевое значение или пустую строку. При подстановке на место логического выражения такие значения преобразуются в False, что приводит к неправильному выводу о том, что свойство не существует. Лучший способ — проверять тип данных свойства, сравнивая его со значением "undefined":

```
if (объект && typeof объект.имяСвойства != "undefined") {
    // Здесь можно работать со свойством
}
```

Ссылку на метод можно получить тем же самым способом, что и ссылку на объект. В условном операторе такая действительная ссылка эквивалентна логическому значению true:

```
if (объект && объект.имяМетода) {
    // Здесь можно работать с методом
}
```

Например, для того чтобы сценарии, написанные в соответствии с правилами адресации W3C DOM, не спотыкались на устаревших браузерах, мож-

но обернуть такие операторы в условие, проверяющее существование метода document.getElementById():

```
function myFucntion() {
    if (document.getElementById) {
        // Здесь можно работать с методом
```

Чтобы сократить размер кода на пару байт, а также избавиться от избыточной вычислительной работы, можно взять глобальные логические переменные-флаги, которые можно использовать в последующих условных операторах:

```
var isW3 = (document.getElementById) ? true:false;

if (isW3) 3
    // Используется свойство
    document.getElementById
}
```

Обсуждение

Оценивая браузер по поддержке объекта или одного из его методов, следует быть осторожным. Например, нельзя считать, что браузер, поддерживающий свойство document.getElementById(), поддерживает и остальные объекты, свойства и методы W3C DOM. Тем не менее по мере накопления опыта вы заметите, что все браузеры, понимающие важнейшие методы, также знают об основных свойствах элементов, таких как nodeType, nodeName и nodeValue. Принять нужное решение вам поможет хорошая справка по DOM.

Не стоит бояться использовать в функциях вложенные проверки существования объектов, свойств и методов. Примеры такого вложения можно увидеть при написании обработчиков для событийной модели IE, несовместимой с W3C (см. рецепт 9.1). Следует всегда удостоверяться в том, что структура функции обеспечивает корректную обработку любого из условий, так что даже при отсутствии необходимого объекта, свойства или метода не происходит ошибок сценария.

Спецификация W3C DOM включает в себя вспомогательные средства, облегчающие определение того, какой уровень поддержки предлагается браузером. Каждый объект, соответствующий элементу документа, имеет метод isSupported(), позволяющий определять поддержку тех или иных стандартов W3C или модулей DOM. Имена модулей, которые можно использовать при проверке поддержки DOM Level 2 таковы: Core, XML, HTML, Views, StypeSheets, CSS, CSS2, Events, UIEvents, MouseEvents, MutationEvents, HTMLEvents, Range и Traversal. Теоретически метод должен возвращать true, если браузер полностью поддерживает нужный модуль:

```
if (myElem.isSupported("CSS2", "2.0")) {
    myElem.style.color = "green":
```

Однако для большинства разработчиков такая проверка недостаточно тщательна. В стандарте есть множество необязательных элементов, отсутствие поддержки которых все равно позволяет браузеру объявлять себя полностью совместимым со стандартом. Единственный способ удостовериться в полной поддержке новых возможностей — использовать явную проверку объектов, свойств и методов.

Смотрите также

Рецепт 5.6 показывает, как проверить поддержку нужного объекта. В рецепте 4.6 описываются специальные значения, которые в условных выражениях эквивалентны true или false. Рецепт 9.1 демонстрирует вложенные проверки объектов в обработчике события.

5.8. Определение основного языка браузера

NN4 IE4

Задача

Необходимо автоматически направить пользователей на ту часть сайта, которая разработана для определенного языка.

Решение

Браузеры 4-й версии и выше предоставляют свойства объекта navigator, позволяющие считывать код языка, для которого разрабатывался браузер.

К сожалению, для разных браузеров это свойство называется по-разному, поэтому необходимо проверять некоторые свойства объекта. В предлагаемом ниже решении все пользователи будут перенаправляться на английскую часть сайта, кроме тех, чьи браузеры используют в качестве основного языка русский:

```
// Проверка основного языка браузера
function getLang(type) {
    var lang;
    if (typeof navigator.userLanguage != "undefined") {
        lang = navigator.userLanguage.toUpperCase();
    } else if (typeof navigator.language != "undefined") {
        lang = navigator language.toUpperCase();
    }
    return (lang && lang.indexOf(type.toUpperCase()) == 0)
}

location.href = (getLang("ru")) ? "ru/home.html"   "en/home.html";
```

Преобразования регистра символов служат здесь для исключения возможных различий в значениях, возвращаемых разными браузерами. В конце функции производится сравнение полученных строк (см. рецепты 1.3 и 1.4).

Обсуждение

Коды языков представляют собой двухбуквенные коды, описывающие основной язык. Кроме того, в них может включаться дополнительный двухбуквенный код, характеризующий страну или регион, для которого адаптирован язык (например, «en-us» для американского английского). Тем самым нельзя полагаться на то, что соответствующее свойство объекта navigator всегда возвращает только два символа. Но так как основной код всегда идет первым, можно проверять только начало хранящейся в свойстве строки. Также следует избавиться от влияния регистра букв, преобразовав все символы к одному виду. Что касается браузеров, не поддерживающих сценарии, то необходимо оставить для их пользователей ссылки, указывающие на нужную часть сайта. Основные двухбуквенные коды языков перечислены в стандарте ISO-639. Выдержка из этого списка доступна по адресу http://www.ietf.org/rfc/rfc1766.txt.

Смотрите также

В рецептах 5.1 и 5.2 демонстрируется определение производителя и версии браузера. В главе 1 описываются процедуры для синтаксического анализа строк.

5.9. Проверка доступности cookie

NN2 IE3

Задача

Сценарий должен определить, разрешен ли пользователем доступ к cookie.

Решение

В Internet Explorer версий 4 и выше, а также в Netscape 6 и выше у объекта navigator имеется свойство cookieEnabled, позволяющее производить такую проверку:

```
if (navigator.cookieEnabled) {
    // Операторы для работы с cookie
}
```

В ранних браузерах можно определять возможность доступа к cookie, проверив сначала существование нужного cookie. Если в нем нет данных, следует попробовать записать в него данные, проверяя, останутся ли они в нем:

```
var cookieEnabled = false;
if (typeof document.cookie == "string") {
    if (document.cookie.length == 0) {
        document.cookie = "test";
        cookieEnabled = (document.cookie = "test");
```

```
        document.cookie = ""
    } else {
        cookieEnabled = true;
```

Обсуждение

Поскольку более сложный способ работает и в тех браузерах, которые предлагают свойство cookieEnabled, можно использовать для всех браузеров одну процедуру для всех поддерживающих сценарии браузеров. В этой процедуре первый оператор if проверяет, содержится ли в свойстве document.cookie строка (что верно и для тех случаев, когда доступ к cookie запрещен пользователем). Если в возвращаемой строке содержится какое-либо значение, можно сделать вывод о том, что cookie доступны для домена, обслуживающего эту страницу. Если же эта строка пуста, делается попытка записать в нее значение, чтобы проверить, удержится ли оно. Если после этого значение может быть прочитано, cookie разрешены. После проверки значение свойства cookie очищается, чтобы не мешать его нормальному использованию.

Смотрите также

Рецепт 1.9 описывает использование cookie. В рецепт 10.4 демонстрируется методика передачи данных между страницами с использованием cookie.

5.10. Формирование ссылок, специфичных для браузера

NN2 **IE3**

Задача

Необходимо создать ссылку, которая указывала бы на разные страницы в зависимости от типа браузера или поддержки некоторых возможностей объектной модели.

Решение

Гиперссылка может указывать на несколько различных документов. При этом обычно один URL жестко устанавливается в атрибуте href тега <a>, а другие ссылки вызываются из сценария, связанного с событием onclick. Обработчик события должен в этом случае предотвратить использование основного URL.

В показанном ниже примере сценарий производит переход к одной из двух страниц: одна — для пользователей Internet Explorer для Windows, другая — для остальных, чьи браузеры поддерживают доступ к элементам с использованием

метода W3C DOM document.getElementById(). Все прочие браузеры перейдут по адресу, указанному в атрибуте href:

```
function linkTo(ieWinUrl, w3Url) {
    var isWin = (navigator.userAgent.indexOf("Win") != -1);
    // Вызов функции, описанной в рецепте 5.3
    var isIE5Min = getIEVersionNumber() >= 5;
    var isW3 = (document.getElementById) ? true : false;
    if (isWin && isIE5Min) {
        location.href = ieWinUrl;
        return false;
    } else if (isW3) {
        location.href = w3Url;
        return false;
    }
    return true
}

<a href="std/newProds.html" title="К новым продуктам"
onclick="return linkTo('ieWin/newProds.html', 'w3/newProds.html')">Новые продукты</a>
```

В том случае, когда сценарий успешно переходит на новую страницу, функция возвращает значение false, благодаря чему обработчик события также возвращает false. Это автоматически отменяет поведение тега <a>, принятое по умолчанию. Если же браузер поддерживает сценарии, но не попадает в одну из двух требуемых категорий, функция возвращает значение true, благодаря чему браузер переходит по адресу, указанному в теге <a>.

Обсуждение

При разработке сайта, который должен нормально восприниматься поисковыми машинами и прочими специализированными программами, следует обязательно снабжать каждый тег <a> действительной ссылкой, даже если ожидается, что все пользователи последуют по адресу, указанному в сценарии. Дело в том, что большинство автоматических служб не интерпретируют JavaScript, а следуют по ссылкам в href.

Еще одна причина заполнять атрибут href для ссылок, которые в действительности отрабатываются с помощью сценария — это текст, который появляется в строке состояния, когда пользователь наводит указатель на ссылку. Существует много различных концепций дизайна пользовательского интерфейса, по-разному относящихся к тому, что должен пользователь видеть в строке состояния. Неопытных пользователей могут привести в замешательство странные имена и косые черточки в этой строке. С другой стороны, опытные пользователи предпочитают видеть полный URL, чтобы знать, что ссылка не уведет туда, куда они не хотят попасть.

Существует способ изменить в обработчиках событий onmouseover и onmouseout текст в строке состояния, появляющийся по умолчанию. Например, в предыдущем примере можно сделать ссылку, выдающую в этой строке сообщение «Наши новейшие продукты...» Такое поведение может быть нормальным для параллельной структуры, на которую намекает показанный пример. Тем не менее опытные

пользователи все равно могут предпочесть видеть реальную ссылку В этом случае можно пойти дальше, используя в обработчиках событий onmouseover и onmouseout функции наподобие linkTo() для того, чтобы выводить текст, зависящий от браузера

Последний полезный совет, который может пригодиться во многих ситуациях, это использование метода location.replace() вместо присвоения значения свойству location.href. Так, в предыдущем примере вместо оператора

```
location.href = ieWinUrl;
```

можно применить следующую запись:

```
location.replace(ieWinUrl);
```

Этот метод перезаписывает содержимое истории браузера так, что новая страница становится на место старой. При этом, щелкнув на кнопке Назад, пользователь не вернется к странице, вызвавшей метод. Судить о том, нужно или не нужно внедрять такую методику, можно по отзывам посетителей сайта.

Смотрите также

Рецепт 10.2 показывает, как сохранить текущую страничку в истории браузера. Рецепт 10.6 показывает, как передавать данные между страничками, используя URL.

5.11. Проверка на разных браузерах

NN2 IE3

Задача

Необходимо проверить работу странички на как можно большом числе версий браузеров различных версий от различных производителей и на разных операционных системах. Однако некоторые браузеры не допускают установку или запуск одновременно нескольких версий.

Решение

Существует коммерческая программа, Virtual PC от Connectix Corporation (http://www.connectix.com), позволяющая запускать на одном компьютере под управлением Windows несколько виртуальных компьютеров. Каждый такой компьютер работает в отдельном окне и не взаимодействует с остальными. Для пользователя Windows это дает возможность создавать и запускать виртуальные компьютеры, на которых установлены разные версии Windows и Internet Explorer. На Macintosh также можно установить несколько виртуальных PC, работающих вместе с Mac OS (рис. 5.1). Благодаря тому, что и виртуальные Windows-машины, и Mac могут использовать общие папки, можно править HTML-файл где угодно, а затем загружать один и тот же файл в несколько версий браузеров одновременно.

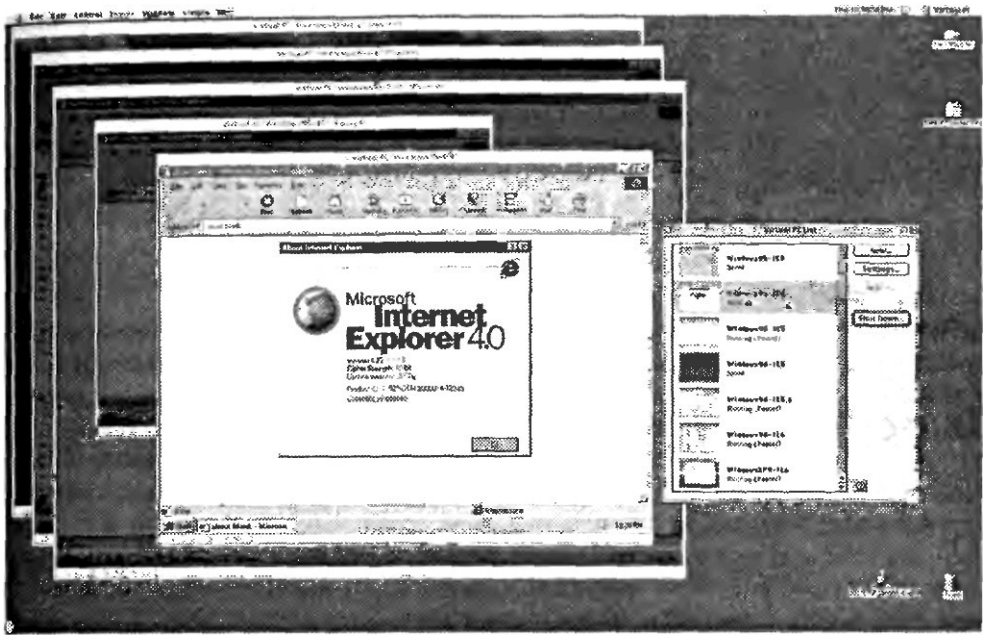


Рис. 5.1. Несколько версий Windows и IE, работающих одновременно на Macintosh

Обсуждение

Поскольку до сих пор нет эмулятора Macintosh, работающего на аппаратно-программной платформе, Wintel, запущенный на Mac, представляет собой более многостороннюю среду, чем на PC. Как под Windows, так и под Mac на Virtual PC можно создавать виртуальные PC под управлением Linux, а также x86 версии Solaris 8 (хотя последняя ОС официально поддерживается только на Virtual PC для Windows).

Устанавливая соответствующие версии ОС и браузеров, можно тем самым создавать машины Windows, являющиеся, по сути, вехами истории. Например, на одной из виртуальных машин может работать Windows 95, Internet Explorer 3, Netscape Navigator 2 и 3. На другой машине могут быть установлены IE 4 и NN4 под управлением Windows 95. На третьей могут быть установлены те же браузеры, но работающие под Windows 98. Можно также установить подходящие версии Opera на соответствующие виртуальные машины (то есть Opera 6 должна быть вместе с IE 6 и Netscape 6).

6 Управление окнами

5.0. Вступление

Пожалуй, наиболее противоречивая чисть DHTML — это путаница с окнами и оконной системой. С одной стороны, окна во многом находятся вне поля зрения DHTML, так как являются просто контейнерами, содержащими страницы. Однако с самых ранних версий управление окнами входило в стандартный пакет, либо предоставляя пользователю дополнительные удобства, либо доставляя ему адские муки из-за огромного количества ненужной чепухи.

Большинство действий, связанных с окнами, используют объект window. За годы развития данный объект получил множество новых свойств и методов, но его реализации в различных браузерах далеки от совместимости. Одна из причин несоответствия между объектами window в различных браузерах состоит в том, что для сценария объект window является контекстом самого верхнего уровня. Такие браузеры, как Internet Explorer для Windows, используют этот объект для хранения многочисленных свойств, связанных с приложением и операционной системой. В противоположность им, Netscape Navigator (особенно с четвертой версии) снабжает этот объект свойствами, потенциально нарушающими конфиденциальность информации пользователя. Эти свойства доступны только для сценариев, пришедших с сервера, которым пользователь непосредственно дал право выполнять такие действия (так называемые подписанные сценарии).

Злоупотребление окнами

К сожалению, некоторые сайты злоупотребляют возможностью автоматически открывать с помощью JavaScript новые окна. В результате появляется устрашающее количество рекламных окон, сильно раздражающих большинство пользователей. Большая часть читателей наверняка хоть раз попадала на сайт с «кошмаром всплывающих окон», когда каждое закрытое окно открывало несколько других. Ситуация начала настолько выходить из-под управления, что некоторые провайдеры выпустили специальные программы, фильтрующие страницы и исключающие их них потенциально опасный код, открывающий вспомогательные окна. Тем не менее, как это часто случается с подобного рода средствами, «блокировщики всплывающих окон» зачастую мешают открытию нужных и безвредных окон. Кроме того, некоторые пользователи вообще отключают JavaScript в сво-

их браузерах, тем самым теряя все преимущества, которые дает им DHTML. В качестве компромиссного решения в некоторых браузерах существует возможность блокировать отдельные действия с окнами.

Блокирование всплывающих окон сейчас настолько распространено, что это должно стать сигналом для разработчика: лучше обойтись одним окном. Если же пользователь захочет открыть новое окно, он сам сможет это сделать с помощью контекстного меню браузера.

Чего нельзя делать

Чем больше окна используются для управления содержимым сайта, тем менее охотно пользователи разрешают подобные действия. Не будь в браузеры встроены значительные средства защиты, существовало бы множество «дыр» как в самих браузерах, так и в интерпретаторах сценариев (некоторые «прорехи» в системе безопасности существовали в старых браузерах, но на сегодняшний день большинство из них исправлены). Поэтому если у вас появится гениальная идея о хитроумном использовании окон, стоит изучить перечень действий, которые нельзя делать даже имея самые добрые намерения.

- **Изменение оформления главного окна.** Несмотря на то что можно изменять размер и положение главного окна браузера (это показано в рецептах 6.1—6.3), нельзя добавлять или убирать элементы его оформления — строку меню, строку состояния, полосы прокрутки, панели инструментов и т. п. В Netscape версии 4 и выше подписанные сценарии могут выполнять подобные действия, но при этом у пользователя всегда запрашивается разрешение. Так что невозможно выполнить эти действия против желания пользователя. Единственный способ управлять оформлением окна — открыть новое окно. Как показано в рецепте 6.4, при открытии нового окна можно выбирать, какие элементы оформления будут в нем присутствовать.
- **Закрытие основного окна из сценария в дочернем.** Если попытаться закрыть главное окно с помощью метода close() (точнее, opener.close()), пользователь увидит предупреждающее сообщение, в котором у него спрашивается разрешение закрыть окно. Такое поведение предотвращает автоматическое закрытие основного окна, которое содержит историю последних посещений. К тому же если дочернее окно не имело заголовка и меню, это могло бы поставить пользователя в неприятное положение.
- **Закрытия окон, открытых другими сценариями.** До сих пор ни в одном из браузеров нет свойства, содержащего массив всех открытых окон. Поэтому если страница открывается в браузере, в котором уже открыто несколько окон, сценарий на ней не сможет ни закрыть их, ни даже узнать их количество.
- **Доступ к свойствам документов в других окнах, обслуживаемых другим доменом.** Было бы весьма неприятно, если сценарий из одного окна мог бы добраться до документа, открытого в другом, и получить из него гиперссылки или какую-либо часть содержимого. В браузерах используется так называемая политика безопасности общего источника, когда сценарий из одного окна (или фрейма) не может получить доступ к критической информации,

хранящейся в другом окне (URL, структуры DOM, содержимому форм и документу), если страницы доставлены с разных серверов, доменов или с использованием разных протоколов.

- Перехват срабатывания или активизация кнопок браузера. Многие авторы сценариев хотели бы перехватывать щелчки на кнопках браузера Вперед и Назад, чтобы управлять навигацией. Но подобные действия не доступны через объект window. В большинстве случаев сценарий может создать эквивалент щелчка на кнопке Печать, но при этом все равно появится диалог с настройками печати, и пользователь должен будет сам щелкнуть в нем на нужных кнопках для начала печати.
- Изменение содержимого поля адреса. Сценарий не может изменить то, как браузер отображает адрес страницы, доставленной с сервера. Единственное, что может сделать сценарий в этом случае, — загрузить с сервера другую страницу (используя свойство location.href), но при этом браузер загрузит и отобразит новую страницу.
- Добавление или удаление записей в меню Избранное/Закладки. Ближе всего можно подойти к этому, используя возможность Internet Explorer для Windows добавлять текущую страницу в список Избранное (используя метод window.external.AddFavorite("URL", "заголовок")). Но даже в этом случае браузер спрашивает разрешения у пользователя внести страницу в избранное. Этот список недоступен для сценариев.
- Изменение настроек браузера. Подписанные сценарии в Netscape 4 и более поздних версиях могут это делать (опять же, с разрешения пользователя). Эта возможность была внедрена в браузер как средство сетевого управления, и не рассчитывалась на использование на сайтах.

Список того, что можно делать с использованием объекта window гораздо длиннее. Внимательно изучая эту информацию, можно заметить, что все перечисленные «табу», которые могут быть весьма удобны в автономных приложениях, могут стать весьма опасными в среде Internet. Уважайте частную жизнь пользователя и его окна.

.1. Управление размером главного окна

NN4 **IE4**

Задача

Необходимо изменить размер окна, содержащего текущую страницу.

Решение

Начиная с четвертой версии Internet Explorer и Netscape Navigator, сценарии имеют возможность изменять размер главного окна браузера в пикселах. Для этого

служат два метода объекта window: resizeTo() и resizeBy(). Чтобы установить определенный размер окна, используйте следующий код:

```
window.resizeTo(800,600);
```

Для увеличения или уменьшения размера окна на определенное количество пикселей служит метод resizeBy():

```
window.resizeBy(50, -10);
```

Эти настройки изменяют размер внешней рамки окна браузера.

Обсуждение

Оба рассмотренных метода имеют два обязательных параметра, измеряемых в пикселах. Первое значение влияет на ширину окна, а второе — на его высоту. Если при использовании метода resizeBy() необходимо оставить одно из измерений неизменным, в качестве другого параметра нужно передать 0.

При изменении размера окна положение его верхнего левого угла остается неизменным. Для того чтобы соответствовать параметрам, переданным методу, изменяют свое положение нижняя и правая границы окна. Если же нужно изменить положение окна или развернуть его, смотрите рецепты 6.2 и 6.3.

Помимо главного окна, эти два метода объекта window могут быть применены также к дочерним окнам, открытым сценарием (см. рецепт 6.4). Для этого воспользуйтесь методами resizeBy() и resizeTo(), принадлежащими объекту нового окна.

Настройка размеров окна в соответствии с содержимым известного размера для разных браузеров не такое простое дело, как может показаться. Только браузеры Netscape 4 и выше имеют пару свойств, innerHeight и innerWidth, позволяющих управлять размером содержимого страницы. В Internet Explorer нет подобных возможностей. С другой стороны, не всегда надежно использовать для задания размеров содержимого размеры внешней рамки окна. Пользователь может установить собственные размеры заголовка окна и панели инструментов, что лишит смысла все аккуратные вычисления, проверенные на тестовых браузерах.

Дизайнеры активно спорят о том, можно ли изменять в сценарии размер уже открытого окна. Большинство пользователей не любит сайты, изменяющих окна их браузеров. Зачастую у пользователей, особенно у опытных, имеется собственные, тщательно сделанные настройки расположения окон приложений на Рабочем столе. И вот появляется «заблудший» сайт, заставляющий окно браузера занять экран целиком. Даже после того, как пользователь покинет сайт, окно сохранит огромный размер, и пользователю придется вручную восстановить вид Рабочего стола. Стоит подумать об этом, прежде чем использовать изменение размера окна.

Смотрите также

В рецепте 6.2 показывается, как установить положение окна на экране. Рецепт 6.3 демонстрирует развертывание окна. Открытию новых окон уделено внимание в рецепте 6.4.

6.2. Перемещение главного окна

NN4 **IE4**

Задача

Необходимо переместить левый верхний угол окна в определенную точку экрана.

Решение

В браузерах, начиная с четвертой версии, имеется два метода объекта window, позволяющих управлять положением окна. Это методы moveTo() и moveBy(). Чтобы передвинуть окно в нужную точку, используется метод moveTo():

```
window.moveTo(10, 20);
```

Для того чтобы сместить положение на нужное количество пикселей по горизонтали и вертикали, применяется метод moveBy():

```
window.moveBy(0, 10);
```

При перемещении окна его размер остается прежним.

Обсуждение

Координатная система расположена на экране так, что левому верхнему углу экрана соответствует точка с координатами (0,0). Видимой области экрана соответствуют положительные координаты по обеим осям. Отрицательные значения, как и значения, превышающие размер экрана, соответствуют невидимой области.

При этом Internet Explorer может передвинуть окно целиком в невидимую область, если это указывается в параметрах, а Netscape Navigator блокирует такие действия. Фактически Netscape Navigator обеспечивает видимость хотя бы малой части окна. Перемещение окна за пределы видимой области экрана — не самая приятная вещь, особенно в Windows, где окно будет оставаться на панели задач, но пользователь не сможет просмотреть его содержимое. В этом случае пользователю придется использовать контекстное меню панели задач для того, чтобы закрыть окно или приложение. Скрытые окна подобного рода впоследствии использовались вместе с некоторыми дырами в системе безопасности для неблаговидных целей, таких как отслеживание действий в других окнах.

Для новых окон, открываемых сценарием, можно указать положение как при их создании (что показано в рецепте 6.4), так и изменять их положение впоследствии. Пока открывший окно сценарий отслеживает ссылку на дочернее окно, можно использовать его методы moveTo() и moveBy().

Смотрите также

Рецепт 6.4 описывает, как изменить размер окна, созданного сценарием.

6.3. Развертывание окна

NN6 **IE5**

Задача

Необходимо изменить размер окна так, чтобы оно заняло всю свободную область окна, как развернутое окно приложения (Windows).

Решение

Для достижения этого результата можно использовать функцию, показанную ниже. Она работает в Internet Explorer 5, Netscape 6 и более поздних версиях:

```
function maximizeWindow() {
    var offset = (navigator.userAgent.indexOf("Mac") != -1 ||
                  navigator.userAgent.indexOf("Gecko") != -1 jj
                  navigator.appName.indexOf("Netscape") != -1) ? 0 • 4:
    window.moveTo(-offset, -offset),
    window.resizeTo(screen.availWidth + (2 * offset).
        screen.availHeight + (2 * offset));
}
```

Несмотря на то что после выполнения окно браузера займет собой весь экран, оно не будет развернуто (в том понимании, которое вкладывается в это слово в Windows).

Обсуждение

Понятие развернутого (и свернутого) окна главным образом используется в Windows. Например, на Macintosh у окон есть значок, позволяющий оптимизировать размер окна приложения. При этом окно располагается так, чтобы остался видимым небольшой участок Рабочего стола. В Windows развернутое окно занимает всю видимую область экрана, исключая панель задач (если она видима).

Сценарий может попытаться только имитировать развернутое окно, и даже при этом имеется несколько ограничений в ряде браузеров. Прежде всего, в Windows настоящее развернутое окно находится не в точке (0,0). Вместо этого левый верхний угол экрана располагается в точке с координатами (-4,-4), то есть немного за пределами экрана. Это позволяет спрятать 4-пиксельную кромку окна и придвинуть его элементы непосредственно к краю экрана. Macintosh ведет себя иначе, оставляя тонкую грань окна видимой.

Но простого определения операционной системы, в которой открыта страница, недостаточно для принятия решения о величине смещения. Netscape (начиная с версии 4) не позволяет располагать окно за пределами экрана, даже если используются подписанные сценарии. Таким образом, применять смещение в четыре пиксела в браузерах, использующих ядро Gecko или носящих имя Netscape, непрактично.

При использовании смещения в четыре пиксела (для IE) необходимо также изменить ширину и высоту окна на удвоенную величину смещения, чтобы окно заняло все доступное пространство. При разворачивании окна таким способом

размер окна браузера становится точно таким же, как и размер обычного развернутого окна. Единственное отличие — значок в правом нижнем углу экрана, позволяющий пользователям вручную изменить размер окна.

При имитации развернутого окна используются свойства объекта screen, availWidth и availHeight, содержащие достаточные для большинства операционных систем сведения. В случае Windows эти свойства содержат размер свободной части экрана, не включающей панель задач. Единственное, чего нельзя определить, — это местоположение панели задач, которую пользователь мог перетащить на другое место. Аналогично, в случае Macintosh свойство screen.availHeight содержит размер области экрана непосредственно под панелью универсального меню. Фактически система координат в браузерах устроена так, что точка с координатами (0,0) соответствует левому верхнему углу свободной области. Следовательно, если поместить окно в координаты (0,0), оно не соскользнет под панель задач, даже если она находится наверху. Чтобы сделать такое «псевдоразворачивание» окна соответствующим Mac, можно изменить ширину, передаваемую методу resizeTo(), так, чтобы оставалось примерно 100 пикселов свободного пространства справа от окна.

Результаты выполнения приведенной в рецепте функции на старых браузерах или браузерах других производителей непредсказуемы. Например, в случае Opera тот факт, что окно официально не развернуто, приводит к тому, что строка заголовка окна остается видимой и влияет на привязку содержимого окна.

Точно так же, как нет способов с помощью сценария развернуть окно, нет способов, чтобы его свернуть. В IE для Windows можно имитировать сворачивание, убирая окно за пределы экрана до тех пор, пока оно не получит фокус (то есть когда пользователь щелкнет на соответствующей ему кнопке на панели задач). Но Windows иногда не посылает окну нужного сообщения, что приводит к невозможности определить, какое из окон владеет фокусом (по крайней мере, так было до появления Windows XP).

Смотрите также

Рецепт 6.1 предлагает советы по изменению размеров окон.

6.4. Создание нового окна

NN2 **IE3**

Задача

Необходимо с помощью сценария открыть отдельное новое окно.

Решение

Для создания нового окна используется метод window.open(), параметрами которого являются: URL-страница, которая будет помещена в окне (абсолютном

или относительном), имя окна и разделенный запятыми список параметров, характеризующих физические характеристики окна. Например:

```
var newWnd = window.open("subDoc.html", "subWindow",
             "status,menubar,height=400,width=300");
```

Если впоследствии сценарий должен будет обращаться к открытому окну, необходимо сохранить в глобальной переменной значение, возвращаемое функцией (ссылку на объект дочернего окна).

Обсуждение

Если не нужно загружать во вновь открытое окно существующие (или формируемые сервером) документы, можно оставить первый параметр функции пустым. При этом откроется пустое окно, которое потом можно будет заполнить динамически. Второй параметр окна, имя, представляет собой значение того же типа, которое указывается в атрибуте target гиперссылки или формы. Следует всегда снабжать открываемые окна уникальными именами, чтобы не путать их друг с другом.

Пожалуй, самый тонкий момент при создании нового окна — это третий параметр метода, содержащий список разделенных запятой свойств окна. Если опустить этот параметр, браузер создаст окно точно такое же, как он создал бы при выборе пункта меню Файл • Создать • Окно (такое окно не обязательно имеет размер главного окна). В более типичных случаях нужно управлять размером, положением и оформлением создаваемого окна. В табл. 1.6 перечислены все атрибуты, которые можно указывать в качестве третьего параметра при создании окна, а также указаны версии браузеров, поддерживающие эти атрибуты.

Таблица 6.1. Атрибуты окна для метода window.open()

Атрибут	NN	IE	Описание
alwaysLowered	4		Окно расположено всегда под другими окнами браузера. Требуется подписанный сценарий
alwaysRaised	4		Окно расположено всегда над другими окнами браузера. Требуется подписанный сценарий
channelMode	—	4	Показывать содержимое на панели каналов
copyhistory	2		Перенести в новое окно историю посещений из текущего
dependent	4		Дочернее окно закроется при закрытии главного
directories	2	3	Показать в новом окне кнопки папок
fullscreen		4	Не отображать заголовок и меню
height	2	3	Размер содержимого окна в пикселах
hotkeys	4		Отключить клавиатурные сокращения для доступа к меню (кроме пунктов Quit и Sequrity Info)
innerHeight	4	—	Высота содержимого. Для установки очень маленьких значений требуется подписанный сценарий

продолжение

Таблица 6.1 (продолжение)

Атрибут	NN	IE	Описание
innerWidth	4		Ширина содержимого. Для установки очень маленьких значений требуется подписанный сценарий
left	6	4	Смещение левой границы окна относительно границы экрана
location	2	3	Отображать поле адреса
menubar	2	3	Отображать строку меню (на Mac строка меню всегда видима, позволяя пользователям менять оформление по желанию)
outer-Height	4	—	Наружная высота окна. Для малых значений требуется подписанный сценарий
outerWidth	4	–	Наружная ширина окна. Для малых значений требуется подписанный сценарий
resizable			Разрешать пользователю менять размер окна (всегда разрешено в Netscape и ранних версиях на Mac)
screenX	4		Смещение левой границы окна от границы экрана. Для установки окна за экраном требуется подписанный сценарий
screenY	4	—	Смещение верхней границы окна от границы экрана. Для установки окна за экраном требуется подписанный сценарий
scrollbars	2	3	Отображать полосы прокрутки для большого документа
status	2	3	Отображать строку состояния
titlebar	4	—	Отображать строку меню. Для того чтобы скрыть меню, нужно записать в него значение по, для чего требуется подписанный сценарий
toolbar	2	3	Отображать панель инструментов
top	6	4	Смещение верхней границы окна относительно границы экрана
width	2	3	Ширина содержимого окна в пикселах
z-lock	4	–	Новое окно фиксируется под прочими окнами браузера. Требуется подписанный сценарий

Можно включать в строку атрибутов параметры, поддерживаемые одними браузерами и неподдерживаемые другими. Если браузер не знает какого-либо из них, он просто игнорирует его. Большая часть атрибутов — булевского типа, указывающие, разрешать ли некоторую опцию в новом окне. Для таких атрибутов можно либо указать числовое значение (yes или 1, чтобы разрешить, и 0 или по для того, чтобы запретить). Для того чтобы разрешить опцию, можно просто указать имя такого атрибута в списке. Следующие два примера оставляют в окне заголовок, строку состояния и позволяют ему менять размер:

```
window.open("someDoc.html", "newWind" "menubar,status,resizable");
window.open("someDoc.html", "newWind" "menubar=1,status=1,resizable=1");
```

По умолчанию булевские атрибуты, управляющие оформлением окна, включены (например, location, resizable, status). Прочие атрибуты выключены (такие

как alwaysRaised и fullscreen). Важно запомнить, что если указать в строке хотя бы один атрибут, все остальные будут автоматически выключены. Таким образом, если вы указываете высоту и ширину окна, необходимо также указать, какие элементы оформления должны в нем присутствовать. Кроме того, при формировании строки атрибутов для большей совместимости следите, чтобы в строке не было пробелов после запятых.

В дополнение к оформлению, появляющемуся в окне, можно указать положение окна на экране. Например, можно с помощью несложных вычислений расположить окно ближе к центру, прежде чем присваивать значения его свойствам width и height (если браузер их поддерживает). Проблема в том, что совместимым между браузерами способом можно управлять только размером содержимого окна, а не его внешней границы. Большие по размеру панели инструментов могут внести погрешность в вычисления. Вот пример функции, открывающей новое окно установленного внутреннего размера по центру экрана:

```
var myWindow:
function openCenteredWindow(url) {
    var width = 400:
    var height = 300:
    var left = parseInt((screen.availWidth/2)    (width/2)).
    var top = parseInt((screen.availHeight/2)    (height/2)):
    var windowFeatures = "width=" + width + ".height=" + height +
        ".status.resizable.left=" + left + ".top=" + top +
        "screenX=" + left + ".screenY=" + top:
    myWindow = window.open(url. "subWind". windowFeatures).
}
```

Если пользователь может открыть окно неоднократно, следует рассмотреть и другие факторы. Смотрите рецепт 6.5 для случая, когда пользователь спрятал окно, и все, что должен сделать сценарий, — вывести его на первый план.

Вопрос о том, можно ли автоматически открывать окна — один из примеров горячо обсуждаемых вопросов дизайна пользовательского интерфейса. К несчастью для программиста, у которого могут быть достойные причины для того, чтобы открыть окно, мир «всплывающей» рекламы настроил многих пользователей против сайтов, вызывающих всплывающие окна. Дочерние окна часто используются для открытия ссылок и отправки форм разработчиками, не желающими терять пользователей. Если бы ссылка на другой сайт открывалась в текущем окне, пользователь мог бы нескоро вернуться на сайт. С другой стороны, результатом такой политики становятся ситуации, когда пользователь завален множеством окон, захламляющих рабочее пространство.

Можно привести массу доводов против использования дочерних окон. Во-первых, пользователи, умеющие применять контекстное меню своего браузера (щелчок правой кнопкой в Windows и Unix, щелчок с удержанием на Mac), могут сами открывать ссылки в новом окне. Это позволяет им самостоятельно контролировать поведение окон. Во-вторых, с точки зрения борцов за чистоту разметки документа, в электронных документах не должны присутствовать дочерние окна и даже фреймы. Например, атрибут target у гиперссылок и форм был убран из стандарта XHTML не случайно. Наконец, третье замечание, которое, пожалуй,

оказывает наибольшее влияние на разработчиков, это борьба, которая ведется в мире со «всплывающей» рекламой. Сейчас множество служб предлагают разнообразные способы блокирования метода window.open() в обслуживаемых или передаваемых ими страницах. Обычно при этом изменяется значение обработчиков событий onload и onunload тега <body>. Но никто не может предсказать, насколько строго провайдеры будут придерживаться этих правил.

Если вы все-таки решите использовать дочерние окна, применяйте их осторожно. Такие окна должны улучшать дизайн или решать какие-либо технические задачи. Чем дольше вы пытаетесь удерживать посетителя с помощью различных трюков, тем меньше шансов, что он вернется на сайт или порекомендует его другим.

Смотрите также

Управление наложением окон описывается в рецепте 6.5. Рецепты 6.6 и 6.7 описывают способы взаимодействия между главным и дочерним окном с помощью сценариев. В рецепте 6.9 показывается переносимая модель модального окна.

6.5. Вывод окна на передний план

NN3 IE4

Задача

Нужно вывести на передний план окно, похороненное под кучей других.

Решение

Для вывода на передний план окна, на которое имеется ссылка, необходимо вызвать его метод focus(). Показанная ниже функция расширяет пример, приведенный в рецепте 6.4. Новая функция не только открывает дочернее окно, но и выводит его на передний план, если оно уже было открыто и находилось под другими окнами.

```
var newWindow;
function makeNewWindow(url) {
    if (!newWindow || newWindow.closed) {
        newWindow = window.open(url,"subwind",
            "status,height=200,width=300");
    } else {
        // Окно уже открыто, поэтому перемещаем его на передний план.
        newWindow.focus();
    }
}
```

Теперь при вызове функции makeNewWindwow() в гиперссылке, при щелчке на кнопки или при другом действии сценария, существующее окно не будет продублировано.

Обсуждение

При загрузке страницы объявленная в показанном выше примере глобальная переменная newWindow инициализируется значением null. Поэтому при первом вызове функции условное выражение в ней истинно, так как в переменной все еще хранится значение null. После того как новое окно создано, в переменную помещается ссылка на соответствующий ему объект. Теперь предположим, что вместо того, чтобы закрыть окно, пользователь щелкнул мышью где-то в главном окне, чтобы вызвать новое окно на поверхность. Если при этом опять вызывается функция makeNewWindow(), первое условное выражение в ней не выполняется (потому что переменная newWindow содержит ссылку на объект), но свойство closed для этого окна имеет значение false. Поэтому выполняется альтернативная ветвь кода, вызывающая у окна метод focus().

В этом примере можно видеть, как важно хранить ссылку на окно, созданное с помощью window.open(). Кроме того, следует внимательно проверять значение свойства closed, прежде чем передавать окну фокус ввода или закрывать его методом close(). Ссылка, один раз помещенная в переменную, никуда не исчезнет при закрытии окна. Переменная будет продолжать хранить то, что ей кажется ссылкой на обычное окно. Но если попытаться использовать такую ссылку, вызвав один из ее методов или обратившись к одному из свойств, произойдет сбой, приводящий к ошибке выполнения сценария. Так как нельзя контролировать, закрыл пользователь окно или нет, сценарий обязан проверять, что происходит.

Смотрите также

В рецепте 6.4 показано, как из сценария открыть несколько окон. Рецепт 6.9 демонстрирует имитацию модального окна.

6.6. Обмен информацией с новыми окнами

NN2 IE3

Задача

Необходимо получить доступ к дочернему окну из главного.

Решение

Если ссылка, возвращенная методом window.open(), сохранена в переменной и содержимое нового окна доставлено с того же сервера, что и содержимое основного, из основного окна можно вызывать те же методы и свойства, которые можно вызывать внутри самого дочернего окна.

Показанная ниже HTML-страница содержит код, создающий новое окно и динамически формирующий его содержимое:

```
<html>
<head>
<title>Новое окно</title>
```

```
<script type="text/javascript">
// Глобальная переменная, хранящая ссылку на окно
var newWindow;
// создаем и заполняем новое окно
function makeNewWindow() {
    // проверка, не открыто ли окно
    if (!newWindow || newWindow.closed) {
        newWindow = window.open("","sub","status,height=200,width=300");
        // задержка перед тем, как окно откроется в IE/Windows
        setTimeout("writeToWindow()", 50);
    } else if (newWindow.focus) {
        // окно уже открыто, выводим его на передний план
        newWindow.focus();
    }
}
function writeToWindow() {
    // формирование содержимого нового окна
    var newContent = "<html><head><title>Дочернее окно</title></head>";
    newContent += "<body><h1>Это окно создано сценарием.</h1>";
    newContent += "</body></html>";
    // запись HTML-кода в новое окно
    newWindow.document.write(newContent);
    newWindow.document.close(); // закрываем поток вывода
}
</script>
</head>
<body>
<h1>Взаимодействие с новым окном</h1>
<hr />
<form>
<input type="button" value="Create New Window" onclick="makeNewWindow();" />
</form>
</body>
</html>
```

Обсуждение

Пример, продемонстрированный в этом рецепте, указывает на важный аспект, который следует учитывать при обращении к недавно созданному окну Internet Explorer для Windows стремится обгонять выполнение сценария, вероятно, с целью повышения производительности. Негативная сторона такой политики — в случае только что созданного объекта ссылка на него может оказаться все еще недействительной в следующем операторе. Чтобы предотвратить ошибки выполнения сценария, которые может вызвать такое опережение, необходимо выделять операторы, обращающиеся к новому объекту, в отдельную функцию, запускающуюся только после того, как поток, выполняющий основную функцию, будет полностью отработан. В решении этой задачи помогает применение функции `setTimeout()`. Насколько большой интервал времени будет указан функции `setTimeout()`, не имеет значения. Задержка в 50 миллисекунд, использованная в примере, представляет собой исключительно короткую паузу (с точки зрения пользователя). Но даже такой задержки достаточно для поддержания правильного порядка выполнения и обеспечения правильности ссылки на новое окно, когда она понадобится. Подобную технику можно использовать во всех случаях,

когда необходимо немедленно обратиться к только что созданному окну. Но если, к примеру, имеется два отдельных действия пользователя (например, две кнопки), одно из них открывает окно, а другое заполняет его данными, использовать задержку не нужно, так как второй обработчик события всегда будет выполняться в отдельном потоке.

Некоторые версии IE для Windows особенно чувствительны к потенциальным нарушениям доступа между доменами. Более того, результаты могут различаться в тех случаях, когда главная страница расположена на жестком диске (при тестировании) или когда она доставляется с сервера (при использовании). О том, что произошла подобная проблема, говорит ошибка выполнения сценария, сообщающая «Access is denied» («Нет доступа»).

Так как ссылка на окно, возвращаемая методом window.open(), является просто ссылкой на объект (то есть не имеет строкового эквивалента), невозможно передавать эту ссылку между страницами, занимающими главное окно. Другими словами, не стоит ожидать, что с дочерним окном, созданным сценарием на одной странице, сможет работать сценарий другой. Единственный способ обойти эту проблему — поместить главную страницу в отдельный фрейм или набор фреймов (при этом прочие фреймы можно спрятать, чтобы пользователь видел только страницу). При создании окна ссылку на него следует поместить в глобальную переменную, в самом наборе фреймов (главном окне) или в одном из фреймов. Новый документ, открытый впоследствии в главном фрейме, при такой организации сможет обратиться к глобальной переменной, чтобы получить ссылку на окно.

К ссылке на дочернее окно нужно относиться так же, как к ссылке на любое другое окно или фрейм. Любые глобальные переменные, объявленные в таком окне, доступны из главного окна:

```
var remoteValue = newWind.someVar;
```

Для доступа к содержимому дочернего окна служит его объект document, как в показанном ниже примере:

```
var remoteBody = newWind.document.body;
newWind.document.getElementById("myTextBox").value = "Фред";
```

Изменять URL в дочернем окне можно точно так же, как и в главном. За исключением того, что следует указывать ссылку на само окно:

```
newWind.location.href = "yetAnoutherPage.html";
```

Тем не менее будьте осторожны при загрузке документов в главное или дочернее окно. Единственный способ закрыть дочернее окно из сценария, не вызвав ошибку, — закрыть его из того же документа, который открыл окно. Кроме того, если сценарий или пользователь загружают в окно страницу с другого сервера или домена, главное окно теряет способность считывать из него значения объекта location и содержимое документа. Такие действия защищают от шпионских сценариев, отслеживающих посещаемые страницы.

Смотрите также

Рецепт 6.7 показывает, как сценарий в дочернем окне может взаимодействовать с главным. Рецепт 10.5 демонстрирует передачу данных через фреймы.

6.7. Обратная связь с главным окном

NN3 IE3

Задача

Необходимо, чтобы сценарий в дочернем окне обращался к данным главного окна.

Решение

За исключением одной ранней версии (Netscape 2), остальные браузеры автоматически создают у дочернего окна, открытого с помощью **window.open(),** свойство opener. Используя это свойство, сценарий из дочернего окна может обращаться к главному. Вот пример сценария в дочернем окне, который копирует значение из поля ввода в скрытый элемент ввода на главном окне:

```
opener.documents.forms["userData"].age.value =
    document.forms["entry"].userAge.value;
```

Свойство opener ссылается на окно или фрейм, в котором был вызван метод open().

Обсуждение

В любом окне, открытом пользователем, свойство opener содержит значение **null.** Следовательно, сценарий может проверять, открыто окно пользователем или сценарием, проверяя значение этого свойства:

```
if (typeof window.opener == "object") {
    // это окно открыто из сценария
}
```

Если дочернее окно было открыто из фрейма, свойство opener содержит ссылку на этот фрейм. Следовательно, из сценария можно обращаться ко всему набору фреймов, если это необходимо. Например, дочернее окно может обращаться к другим фреймам, используя следующий синтаксис:

```
opener.parent.frames["prefs"].document.dataForm.colorChoise.value =
    "#66eeff";
```

Политика общего источника, используемая при управлении доступом к дочернему окну (см. рецепт 6.6), применяется и в обратном направлении. Если документ в главном окне или фрейме сменился документом с другого сервера или домена, попытка обратиться к нему из дочернего окна приведет к отказе в доступе.

Смотрите также

В рецепте 6.6 можно увидеть, как сценарий в главном окне может взаимодействовать с содержимым дочернего.

6.8. Модальные и немодальные окна IE

IE4

Задача

Необходимо приостановить выполнения сценария на то время, пока открыто модальное диалоговое окно, а затем получить из него введенные пользователем значения.

Решение

В Internet Explorer версий 4 и выше (как для Windows, так и для Macintosh) у объекта window имеется метод, позволяющий открывать настоящие модальные диалоговые окна (предотвращающие доступ к главному окну до тех пор, пока диалог не будет закрыт). В Internet Explorer 5 для Windows и более поздних версиях также имеется способ создать окно, которое бы всегда находилось перед другими окнами браузера, но не ограничивало бы доступ к элементам интерфейса главного окна. Эти методы называются window.showModalDialog() и window.showModelessDialog() соответственно.

Для того чтобы использовать оба этих метода, следует сначала объединить данные, передаваемые диалогу в один объект JavaScript любого типа (если такие данные нужны). Здесь для этого служит переменная dialogArgs. Затем необходимо определить место в сценарии, где нужно запросить у пользователя параметры, и вызвать метод:

```
var dialogAnswer = window.showModalDialog("dialog.html", dialogArgs,
    "dialogWidth:300px: dialogeight:201 px: center:yes");
```

Сценарий в документе, загружаемом в диалоговое окно, может использовать для доступа к переданным аргументам свойство window.dialogArguments. Для того чтобы вернуть значения обратно в основное окно, следует присвоить эти значения (любого типа) свойству window.returnValue. После того как пользователь закроет модальное диалоговое окно, это значение будет присвоено переменной, стоящей в левой стороне вызывающего окно оператора В данном примере это переменная dialogAnswer.

Обсуждение

Модальные диалоговые окна в IE не поддерживают связь того же рода, что использовалась с обычными дочерними окнами, созданными с помощью window.open(). Но, с другой стороны, связь между диалогом и главным окном не разорвана полностью.

Например, сценарий в главном окне может передать методу showModalDialog() ссылку на любой элемент документа главного окна. Впоследствии сценарий в диалоговом окне может взять переданную ссылку для доступа к свойствам

этого объекта. Вот простой пример, начинающийся с главного окна, передающего модальному диалоговому окну ссылку на форму (элемент form):

```
<html>
<head>
<title>Откройте диалоговое окно</title>
<script type="text/javascript">
function openDialog(form) {
    var result = window.showModalDialog("dialogDoc.html", form.
        "dialogWidth:300px; dialogHeight:201px; center:yes");
}
</script>
</head>
<body>
<h1>Модальное диалоговое окно Internet Explorer</h1>
<hr />
<form name="sample" action="#" onsubmit="return false">
Введите свое имя:<input name="yourName" type="text" />
<input type="button" value="Send to Dialog" onclick="openDialog(this.form)" />
</form>
</body>
</html>
```

При этом сценарий в диалоговом окне может считывать значения из формы в главном:

```
<html>
<head>
<title>Модальное окно</title>
</head>
<body>
<script type="text/javascript">
document.write("Поздравления от " + window.dialogArguments.yourName.value + "!");
</script>
</body>
</html>
```

Немодалыюе окно с точки зрения написания сценария ведет себя несколько иначе. Главное отличие в том, что выполнение сценария в основном окне не останавливается при открытии такого диалога. Это вполне логично, поскольку немодальный диалог должен обеспечивать пользователю возможность взаимодействия с обоими окнами, в то время как диалог просто остается на переднем плане. Второе отличие в том, что значение, возвращаемое методом showModelessDilog(), представляет собой ссылку на созданное диалоговое окно. Это позволяет основному сценарию взаимодействовать с ним после создания.

Нередко при вызове метода showModelessDialog() ему передается ссылка либо на главное окно, либо на одну из функций, принадлежащих ему. Такая функция впоследствии должна будет вызвана из сценария в диалоговом окне (тут используется та же идея, что у кнопки Применить, имеющейся на многих диалоговых окнах в Windows. Передача ссылки на главное окно будет выглядеть так:

```
var dialogWind = window.ShowModelessDialod("myModeless.html", window.
    "dialogWidth:300px; dialogHeight:201px, center:yes");
```

Поле этого сценарий в диалоговом окне может использовать ссылку window.dialogArguments для доступа к любым глобальным переменным, функциям или элементам документа в главном окне:

```
var mainWnd = window.dialogArguments;
mainWind.doument.body.style.backgroundColor = "lightYellow";
```

Еще одно отличие немодальных диалогов в том, что в них не применяется свойство window.returnValue, то есть обмениваться информацией с главным окном необходимо напрямую. Один из способов обмена информацией — передать в диалог ссылку на главное окно. При этом для вызова глобальной функции можно использовать такой синтаксис:

```
var mainWind = window.dialogArguments;
mainWind.myFunction();
```

Или же можно передать в диалог ссылку на одну из функций в главном окне:

```
# в главном сценарии
window.showModelesDialog("myModeles.html", myFunction, '...');

# в сценарии диалогового окна
var mainFunc = window.dialogArguments;
mainFunc();
```

Необязательный третий параметр, использующийся при создании любого из типов диалоговых окон, представляет собой разделенный запятыми список характеристик окна диалога. Синтаксис этого параметра, как можно видеть в предыдущих примерах, напоминает синтаксис CSS, имя:свойство. В табл. 6.2 перечислены имена свойств и приведено их описание.

Таблица 6.2. Свойства для методов showModalDialog() и showModelessDialog()

Свойство	Значение	По умолчанию	Описание
center	yes | no j 1 | 0 | on j off	yes	Располагать диалог по центру
dialogHeight	Размер в пунктах	—	Высота окна диалога (для IE/Mac должна быть не меньше 200)
dialogLeft	Целое число	—	Число пикселей слева (отменяет center)
dialogTop	Целое число	—	Число пикселей сверху (отменяет center)
dialogWidth	Размер в пунктах	—	Ширина окна диалога (для IE/Mac должна быть не меньше 200)
edge	raised | sunken	raised	Стиль перехода между границей окна и содержимым
help	yes | no | 1 | 0 | on | off	yes	Отображать значок справки в заголовке окна
resizable	yes | no | 1 | 0 | on | off	no	Изменяемый размер диалога
status	yes | no | 1 | 0 | on | off	yes	Отображать строку состояния

СОВЕТ

Модальные и немодальные диалоговые окна, как и любой другой потенциально назойливый элемент интерфейса, следует использовать с осторожностью.

Смотрите также

В рецепте 6.9 показано, как создать модальное или немодальное окно в IE и Netscape, используя механизм дочерних окон. В рецепте 6.10 описана имитация модального диалогового окна с использованием слоев.

6.9. Имитация совместимого модального диалога

NN4 IE4

Задача

Необходимо сделать устойчивое окно, работающее на разных браузерах.

Решение

Метод showModalDialog(), предлагаемый Internet Explorer, не поддерживается ни одним другим браузером. В этом рецепте показано, как использовать дочернее окно для того, чтобы имитировать модальный диалог. Этот рецепт работает в IE 4, Netscape 4, Opera 6 и более старших версиях этих браузеров. Из-за некоторых странностей в поведении Internet Explorer для Windows определенные пользователи могут обойти «модальность» такого диалогового окна и активизировать ссылки в главном окне браузера. Впрочем, для большинства пользователей это окно неотличимо от обычного модального диалога.

В главную страницу необходимо встроить предлагаемую в секции обсуждения библиотеку simModal.js. Эта библиотека блокирует доступ к элементам форм и гиперссылкам главного окна, а также следит за тем, чтобы фокус оставался в диалоговом окне, тем самым заставляя пользователя работать именно с ним. После того как диалог закрывается, доступ к элементам главного окна вновь открывается.

Ниже приведен скелет HTML-страницы, в которой показаны обработчики событий, необходимые для работы библиотеки, и пример того, как с ее помощью открыть модальное диалоговое окно. В данном примере это диалог с настройками:

```
<html>
<head>
<title>Main Application Page</title>
<script type="text/javascript" src="simModal.js"></script>
<script language="JavaScript" type="text/javascript">
// Функция, выполняемая после того, как в диалоги нажата кнопка "OK".
function setPrefsO {
    // Возвращенное значение просто показывается в текстовом поле
```

```
    document.returned.searchURL.value = dialogWin.returnedValue;
}
</script>
</head>
<body onclick="checkModal()" onfocus="return checkModal()">
<!-Здесь должно быть расположено содержимое страницы ->
<a href="noPrefs.html"
    onmouseover = "status = 'Set preferences..  ;return true"
    onmouseout = "status='';return true."
    onclick = "openSimDialog('dialog_main.html' 400. 300. setPrefs),return false">
Настройки
</a>
<!-Здесь должно быть расположено содержимое страницы ->
</body>
</html>
```

для применения библиотеки следует добавить в тег <body> обработчики событий onclick и onfocus, как это показано в примере. Эти обработчики для проверки того, что фокус удерживается в диалоговом окне, вызывают внешнюю функцию checkModal(), определенную в библиотеке. Для вызова диалогового окна служит функция openSimModalDialog(), в качестве параметров получающая URL к странице, отображаемой диалоговым окном, размеры окна (в пикселах), а также ссылку на функцию в главном окне, вызываемую при закрытии диалога. В данном случае это функция setPrefs().

На странице, отображаемой в диалоговом окне, нужно создать три функции: closeme(), handleOK() и handleCancel(), используемые в показанной ниже выдержке для обработки нажатий на кнопки OK и Cancel. Обработчики событий onload и onunload в теге <body> управляют блокировкой событий и связаны с библиотечными функциями blockEvents() и unblockEvents().

```
<html>
<head>
<title>Настройки</title>
<script language="JavaScript">
// закрытие диалога
function closeme() {
    window.close();
}

// Обработка нажатия кнопки DK
function handleDKO {
    if (opener && !opener.closed) {
        top.dlogBody.transferData();
        opener.dialogWin.returnFunc();
    } else {
        alert("Главное окно было закрыто.\n\настройки  сделанные в этом" +
              "диалоге. не будут использованы.");
    }
    closemeO;
    return false;
}

// Обработка нажатия кнопки Cancel (Отмена)
function handleCancel() {
    closemeO;
```

```
    return false.
}
</script>
</head>
<body onload="if (opener && opener. block Events) opener. blockEventsO"
    onunload="if (opener && opener.unblockEvents" opener.unblockEvents()">
<!- Тут должно быть расположено содержимое диалога ->
<form>
<input type="button" value="Cancel" onclick="handleCancel()">
<input type="button" value-"  OK  " onclick="handleOK()">
</form>

</body>
</html>
```

Если диалоговое окно содержит набор фреймов, обработчики событий onload и onunload необходимо расположить в теге <frameset> В этом же документе следует разместить все три функции, а вызывать их из обработчиков нажатия кнопок так: parent.handleOK() и parent.handleCancel().

Обсуждение

В листинге 6.1 приведена библиотека simModal.js, которую необходимо присоединить к основной странице.

Листинг 6.1. Библиотека имитации модальных окон (simModal.js)

```
// Глобальный флаг для запуска кода, использующегося только на Netscape 4
var Nav4 = ((navigator.appName ==="Netscape") && (parseInt(navigator.appVersion) -= 4))

// Объект, отслеживающий модальный диалог, открытый из этого окна
var dialogWin = new Object();

// так как в некоторых браузерах ссылки не могут быть полностью блокированы.
// убираем обработчики событий onclick и onmouseout
// Восстанавливаем их, снимая блокировку с главного окна
var linkClicks;

// Обработчик события, блокирующий формы в Navigator 4
// и гиперссылки в IE, когда активен диалог
function deadendO {
    if (dialogWin.win && !dialogWin.win.closed) {
        dialogWin.win.focus();
        return false;



// Блокирование форм и гиперссылок во всех фреймах
function disableForms() {
    linkClicks - new Array();
    for (var i = 0; i < document.forms.length; i++) {
        for (var j = 0; j < document.forms[i].elements.length; j++) {
            document.forms[i].elements[j].disabled = true:
        }
    }
    for (i = 0; 1 < document links.length; i++) {
```

```
        linkClicks[i] = {click:document.links[i] onclick, up:null};
        linkClicks[i].up = document.links[i].onmouseup;
        document.links[i].onclick = deadend;
        document.links[i].onmouseup = deadend;
        document.links[i].disabled = true;
    }
    window.onfocus = checkModal;
    document.onclick = checkModal;
}
// Восстановление нормального поведения форм и гиперссылок.
function enableForms() {
    for (var i = 0; i < document.forms.length; i++) {
        for (var j = 0; j < document.forms[i].elements.length; j++) {
            document.forms[i].elements[j].disabled = false;
        }
    }
    for (i = 0; i < document.links.length; i++) {
        document.links[i].onclick = linkClicks[i].click;
        document.links[i].onmouseup = linkClicks[i].up;
        document.links[i].disabled = false;
    }
}

// Перехватываем все события в Navigator, коорые могут прийти из элементов
// формы, когда диалог открыт. В случае IE элементы форм блокируются.
function blockEvents() {
    if (Nav4) {
        window.captureEvents(Event.CLICK | Event.MOUSEDDWN | Event.MOUSEUP | Event.FDCUS);
        window.onclick = deadend;
    } else {
        disableForms();
    }
    window.onfocus = checkModal;
}

// При закрытии диалога восстанавливаем нормальное поведение окна
function unblockEvents() {
    if (Nav4) {
        window.releaseEvents(Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP | Event.FOCUS);
        window.onclick = null;
        window.onfocus = null;
    } else {
        enableFormsO;
```

```
// Создание модального диалогового окна
// Параметры
//    url - URL страницы или набора фреймов, загружаемых в окно.
//    width - ширина диалогового окна в пикселах
//    height - высота диалогового окна в пикселах
//    returnFunc - ссылка на функцию (на этой странице).
Ч                 обрабатывающую данные, возвращаемые из диалога
//    args - [необязательный] любые данные, которые нужно передать в диалог
```

продолжение

Листинг 6.1 (продолжение)

```
function openSimDialog(url, width, height, returnFunc, args) {
    if (!dialogWin.win || (dialogWin.win && dialogWin.win.closed)) {
        // Начальная установка свойств нового диалога
        dialogWin.url = url;
        dialogWin.width = width;
        dialogWin.height = height;
        dialogWin.returnFunc = returnFunc;
        dialogWin.args = args;
        dialogWin.returnedValue = "";
        // Следим за тем, чтобы имя было уникальным.
        dialogWin.name = (new Date()).getSeconds().toString();
        // Сборка атрибутов окна. Пытаемся расположить диалог по центру
        if (window.screenX) {              // Netscape 4+
            // Располагаем по центру главного окна
            dialogWin.left = window.screenX +
               ((window.outerWidth - dialogWin.width) / 2);
            dialogWin.top = window.screenY +
               ((window.outerHeight - dialogWin.height) / 2);
            var attr = "screenX=" + dialogWin.left +
               ",screenY=" + dialogWin.top + ",resizable=no,width=" +
               dialogWin.width + ",height=" + dialogWin.height;
        } else if (window.screenLeft) {    // IE 5+/Windows
            // Располагаем более-менее по центру главного окна IE.
            // Сначала оцениваем размер окна.
            // Принимаем в расчет режим совместимости IE6+ и CSS
            var CSSCompat = (document.compatMode && document.compatMode != "BackCompat");
            window.outerWidth = (CSSCompat) ? document.body.parentElement.clientWidth
document.body.clientWidth;
            window.outerHeight = (CSSCompat) ? document.body.parentElement.clientHeight :
document.body.clientHeight;
            window.outerHeight -= 80;
            dialogWin.left = parseInt(window.screenLeft+
               ((window.outerWidth - dialogWin.width) / 2));
            dialogWin.top = parseInt(window.screenTop +
               ((window.outerHeight - dialogWin.height) / 2));
            var attr = "left=" + dialogWin.left +
               ",top=" + dialogWin.top + ",resizable=no,width=" +
               dialogWin.width + ",height=" + dialogWin.height;
        } else {                           // all the rest
            // Лучшее, что можно сделать. - расположить по центру экрана.
            dialogWin.left = (screen.width - dialogWin.width) / 2;
            dialogWin.top = (screen.height - dialogWin.height) / 2;
            var attr = "left=" + dialogWin.left + ",top=" +
               dialogWin.top + ",resizable=no,width=" + dialogWin.width +
               ",height=" + dialogWin.height;
        }

        // Формируем диалог и устанавливаем на него фокус.
        dialogWin.win=window.open(dialogWin.url, dialogWin.name, attr);
        dialogWin.win.focus();
    } else {
        dialogWin.win.focus();
    }
```

```
// Вызывается обработчиком события КАЖДОГО фрейма.
// возвращая фокус в диалог, если он открыт
function checkModal() {
    setTimeout("finishChecking()" 50):
    return true:
}

function finishChecking() {
    if (dialogWin.win && !dialogWin.win.closed) {
        dialogWin.win.focus();
    }
>
```

Код библиотеки начинается с объявления нескольких глобальных переменных, переносящихся через все приложение. Одна из переменных, флаг Nav4, предназначена только для Navigator 4, другая, dialogWin, содержит ссылку на диалоговое окно.

Функция deadend() представляет собой обработчик события. Этот обработчик библиотека назначает всем гиперссылкам главного окна, пока активен диалог. Эта функция блокирует срабатывание ссылки при щелчке на ней, а также блокирует все относящиеся к мыши события в Netscape 4.

Далее расположена пара функций, одна из которых блокирует все элементы форм и гиперссылки главного окна, а другая — восстанавливает доступ к ним. Метод disableForms() обязательно вызывается при открытии диалогового окна, (фактически эту функцию вызывает обработчик события onload, запускающий функцию blockEvents(), которая, в свою очередь, вызывает disableForms()). Исходные значения обработчиков событий для гиперссылок сохраняются на время блокирования в переменной linkClicks, в то время как их заменяет вызов функции deadend(). Когда модальное окно закрывается, функция enableForms() восстанавливает их исходное значение.

Действия, выполняемые функцией blockEvents(), несколько различаются на разных браузерах. Механизм перехвата событий в Netscape 4 решает большинство проблем, в то время как в остальных браузерах следует использовать функцию disableForms(). Когда придет пора восстановить все как прежде, функция unblockEvents(), вызываемая в обработчике события onunload, обращает этот процесс.

Сердце всей библиотеки — функция openSimDialog(). Эта функция получает несколько параметров, позволяющих указать URL страницы, которая будет загружена в окно, размеры диалогового окна, ссылку на функцию главного окна и необязательные параметры, передаваемые в диалог (хотя традиционные способы взаимодействия с дочерними окнами остаются в силе, см. рецепты 6.6 и 6.7). Большая часть кода этой функции посвящена вычислению (иногда приблизительному) координат нового окна так, чтобы оно располагалось по центру основного окна. Эта функция также заполняет глобальный объект dialogWin, назначение которого — хранение важных значений, необходимых для сценария диалогового окна.

После установочного кода идут две функции, checkModal() и связанная с ней finishCheckingO, которые заставляют дочернее окно вести себя как модальный диалог, возвращая ему фукус, если на передний план попытается выйти главное окно. Тайм-аут нужен для решения обычных проблем с синхронизацией окон, относящихся по большей части к IE для Windows.

Имитированное модальное диалоговое окно — довольно сложное приложение JavaScript. Этот сценарий не столько заботится о придании модальности диалоговым окнам в Netscape, сколько пытается обойти ошибку в ранних версиях Internet Explorer, из-за которой было невозможно применять в диалоговых окнах фреймы. С использованием обычных окон эта проблема может быть решена, и после небольшой отладки решение работает на Netscape и на Opera. Более ранние версии этого сценария появились в статье на сайте для Netscape разработчиков.

Одно из важных различий между предложенным решением и функцией showModalDialog() в IE заключается в том, что выполнение сценария в главном окне не прерывается при открытии диалога. Здесь имитация модального диалога ведет себя во многом как showModelessDialog() в IE. Обратите внимание на то, что некоторые из аргументов функции openSimDialog() помещаются в свойства глобального объекта dialogWin. Этот объект работает как склад ключевых данных об окне, включая саму ссылку на объект окна (свойство dialogWin.win). Другое из этих свойств, returnFunc, представляет собой ссылку на одну из функций в главном окне, которая будет вызвана при закрытии дочернего окна. Впрочем, сценарий в диалоговом окне имеет возможность вызвать эту функцию в любой момент времени. Эта функция служит для организации нужной последовательности выполнения сценария, наподобие того, как это происходит при использовании настоящих модальных окон. Вызвать ее очень просто:

```
opener.dialogWin.returnFunc();
```

Если необходимо возвратить из диалога значения, это можно сделать, например, так:

```
opener.dialogWin returnFunc(docment myForm.myTextBox.value);
```

Для передачи данных в диалоговое окно служит необязательный пятый параметр функции openSimDialog(). Несколько значений можно передать, объединив их в объект или массив. Сценарий в диалоговом окне может получить переданное значение с помощью свойства dialogWin.args. Например:

```
var passedValue = dialogWin.args:
```

Обычно диалоговое окно запрашивает у пользователя некоторые данные, которые затем оказывают влияние на главное окно или его данные. Хороший тон в программировании пользовательского интерфейса — всегда предоставлять пользователю возможность вернуться из диалогового окна, не внося никаких изменений. В данном примере показано, как с помощью двух кнопок (или их эквивалентов) можно дать пользователю выбор — принять внесенные изменения (OK) или отказаться от них (Cancel). Кроме того, показанный код проверяет, не закрыл ли пользователь главное окно (так как сценарий не может заблокировать кнопку закрытия окна).

Использование имитации модального окна в том случае, когда главное окно содержит набор фреймов, немного сложнее, но тоже возможно. Ключ к успешной реализации — перемещение функций enableForms() и disableForms() (а также вспомогательных функций) в набор фреймов. Обе функции следует модифицировать, чтобы они блокировали или разблокировали все элементы всех фреймов набора Для сохранения значений можно продолжать использовать переменную

linkClicks, но уже как массив из массивов, каждый элемент которого будет представлять массив, соответствующий отдельному фрейму. Вот пример того, как следует изменить функцию disableForms():

```
//Блокирование элементов форм и гиперссылок во всех фреймах.
function disableForms() {
    linkClicks = new Array();
    for (var h = 0; h < frames.length; h++) {
        for (var i = 0; i < frames[h].document.forms.length; i++) {
            for (var j = 0; j < frames[h].document.forms[i].elements.length;
                j++) {
                frames[h].document.forms[i].elements[j].disabled = true;
            }
        }
        linkClicks[h] = new Array();
        for (i = 0; i < frames[h].document.links.length; i++) {
          if (typeof frames[h].document.links[i].disabled != "undefined"){
          alert(i)
              frames[h].document.links[i].disabled = true;
          } else {
                linkClicks[h][i] = frames[h].document.links[i].onclick;
                frames[h].document.links[i].onclick = deadend;


        frames[h].window.onfocus = checkModal;
        frames[h].document.onclick = checkModal;



// Восстановление нормального поведения всех элементов форм и гиперссылок.
function enableForms() {
    for (var h = 0; h < frames.length; h++) {
        for (var i = 0; i < frames[h].document.forms.length; i++) {
            for (var j = 0; j < frames[h].document.forms[i].elements.length;
                j++) {
                frames[h].document.forms[i].elements[j].disabled = false;


        for (i = 0; i < frames[h].document.links.length; i++) {
          if (typeof frames[h].document.links[i].disabled != "undefined")}
              frames[h].document.links[i].disabled = false;
          } else {
                frames[h].document.links[i].onclick = linkClicks[h][i];
```

Смотрите также

В рецепте 6.8 описывается специфическая для IE (и более ясная) реализация модальных и немодальных диалоговых окон. Рецепт 6.10 демонстрирует использование слоев для имитации диалоговых окон. В рецептах 3.1 и 3.7 обсуждается использование массивов и пользовательских объектов, которые можно применять для передачи данных в диалоговые окна.

6.10. Имитация окон с помощью слоев

NN6 IE5

Задача

Необходимо создать эффект отдельного, перетаскиваемого мышкой окна, в действительности не создавая окон.

Решение

Предлагаемое решение состоит из нескольких отдельных файлов, в том числе оно включает в себя три файла .js библиотек JavaScript и несколько таблиц стилей .css. Один из библиотечных файлов, DHTML2API.js, взят прямо из рецепта 13.3, два других, layerDialog.js и LayerDialogDrag.js, которые служат соответственно для создания диалогового окна и его перемещения с помощью мыши, приведены в секции обсуждения. Файлы .css придают создаваемым окнам разный вид на разных системах и также приведены в обсуждении.

Ниже приведена базовая HTML-страница, в которой показано, как присоединить нужные библиотеки и как настроить необходимые элементы (несколько элементов div, span и iframe), составляющие псевдоокно. Также указано, как сделать гиперссылку, щелчок на которой открывает немодальное, перетаскиваемое мышью диалоговое окно (в данном случае это окно настроек).

```
<html>
<head>
<title>Main Application Page</title>
<link rel="stylesheet" id="mainStyle" href=" ./css/cookbook.css" type="text/css" />
<script type="text/javascript" src=". ./js/DHTMLAPI.js"></script>
<script type="text/javascript" src="layerDialog.js"></script>
<script type="text/javascript" src="layerDialogDrag.js"></script>
<script type="text/javascript">
// Функция. которая будет запущена после нажатия кнопки "OK".
function setPrefsO {
    // Операторы, применяющие к странице введенные пользователем настройки
}
</script>
</head>
<body onload="initDHTMLAPI(); initDrag(); initLayerDialog()">
<hl>Giantco. Inc.</hl>
<a href="noPrefs.html" onmouseover="status='Настройки...';return true"
            onmouseout="status='';return true"
            onclick="openLayerDi a log ('dialog_main.html''Настройки', setPrefs. null);return
false">
Настройки
</a>
<!-Тут расположен остальной текст страницы ->
<div id="pseudoWindow">

<div id="titlebar" class="draggable"><img id-"closebox"
```

```
src="closeBox_win9x.jpg" onclick="closeLayerDialog()" />
<span id="barTitle">Заголовок</span></div>

<iframe id="contentFrame" src="" frameborder="0" vspace="0" hspace="0"
marginwidth="14" marginHeight="14" width="100%" height="480" scrolling="auto">
</iframe>

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

Три ключевые подпрограммы инициализации, initDHTMLAPI() (из библиотеки DHTML2API.js), **initDrag()** (из библиотеки **layerDialogDrag.js**) и initLayerDialog() (библиотека LayerDialog.js) вызываются в обработчике события onload тега <body>.

Обсуждение

Настройка внешнего вида подвижных элементов, используемых в качестве имитации диалоговых окон, становится несколько запутанной, если приходится учитывать огромное разнообразие того, как могут выглядеть популярные операционные системы. Окно, оформленное в стиле Windows 98, будет совершенно не похоже на окно в Windows XP. В мире Mac традиционное оформление окон в Mac OS 9 также разительно отличается от оформления окон в Mac OS X. В данном рецепте принимаются следующие предположения о диалоговых окнах, составленных из подвижных элементов:

- окно имеет фиксированный размер, управляемый из таблиц стилей (в примере используется размер 800x600);
- заголовок окна и вид закрывающей кнопки настраиваются в соответствии с одной из операционных систем: Windows 9x, Windows XP, Mac OS и Mac OS X;
- во всех остальных операционных системах диалоговое окно выглядит как в Windows 9x;
- содержимое окна может загружаться с любого URL;
- окно можно перетаскивать мышью, оно ведет себя как немодальное диалоговое окно.

Основанное на HTML псевдоокно представляет собой составной перетаскиваемый мышкой объект. Содержимое окна заключено в тег с идентификатором **pseudoWindow**. В него вложены два элемента. Первый из них представляет собой элемент div, формирующий строку заголовка окна. У этого элемента есть два дочерних узла: изображение, соответствующее закрывающей кнопке окна, и элемент span, содержащий текст заголовка. Второй узел главного элемента **div** — элемент **iframe**, содержащий страницу с диалоговым окном.

Основой интерактивности, которой обладает окно, являются три библиотеки JavaScript: DHTML2API.js, layerDialog.js и layerDialogDrag.js. Библиотека **DHTML2API.js** содержит множество функций для позиционирования элементов, работающих на многих браузерах, и используется для позиционирования окна диалога.

В листинге 6.2 показан код библиотеки layerDialog.js, выполняющей две важные задачи: присоединение к странице таблицы стилей, специфичной для опера-

ционной системы пользователя, и управление созданием, начальным расположением и выводом псевдоокна на экран.

Листинг 6.2. Библиотека layerDialog.js

```
// Выбор одного из вариантов оформления псевдоокна
function getCurrDSUI() {
    var ua = navigator.userAgent;
    if (ua.indexOf("Mac") != -1) {
        if(ua.indexOf("OS X") != -1 || ua.indexOf("MSIE 5.2") != -1) {
            return "macosX"
        } else {
            return "macos9";
        }
    }else if (ua.indexOf("Windows XP") != -1 j| ua.indexOf("NT 5.1") != -1){
        return "winxp";
    } else if ((document.compatMode && document.compatMode != "BackComp") j|
        (navigator.product && navigator.product == "Gecko")) {
        // Win9x, совместимая с CSS
        return "win9x";
    } else {
        // Значение по умолчанию, использующееся для Win9x при странных
        // настройках, Unix/Linux и для неизвестных ОС
        return "wi n9xQ";
    }
}
var currOS - getCurrOSUI();
// Загрузка таблицы стилей, специфической для ОС пользователя
document.write("<link rel='stylesheet' type='text/css' href='dialogLayer_" + currOS +
".css'>");

//******************************
//  НАЧАЛО КОДА ДИАЛОГА
//******************************/

// объект, отслеживающий текущее псевдоокно.
var dialogLayer = {layer:null, visible:false};

// Центруем подвижный элемент, имя которого передано в качестве параметра
// по отношению к текущему окну или форме, и показываем его функцию centerOnWindow(elemID) {
    // 'obj' является перемещаемым элементом
    var obj = getRawObject(elemID);
    // параметры прокрутки окна
    var scrollX = 0, scrollY = 0;
    if (document.body && typeof document.body.scrollTop != "undefined") {
        scrollX += document.body.scroll Left;
        scrollY += document.body.scrollTop;
        if (document.body.parentNode &&
            typeof document.body.parentNode.scrollTop != "undefined") {
            scrollX += document.body.parentNode.scrollLeft;
            scrollY += document.body.parentNode.scrollTop
        }
    } else if (typeof window.pageXOffset != "undefined") {
```

```
        scrollX += window.pageXOffset;
        scrollY += window.pageYOffset;

    var x = Math.round((getInsideWindowWidth()/2) -
        (getObjectWidth(obj)/2)) + scrollX;
    var y = Math.round((getInsideWindowHeight()/2)
        (getObjectHeight(obj)/2)) + scrollY;
    shiftTo(obj, x, y);


function initLayerDialog() {
    document.getElementById("closebox").src = "closeBox_" + currOS + ".jpg"
    dialogLayer.layer = document.getElementById("pseudoWindow");
}

// Установка и вывод псевдоокна.
// Аргументы:
//    url - URL страницы или фрейма, загружаемого в iframe
//    returnFunc - ссылка на функцию (на этой странице).
//                 обрабатывающую данные, возвращенные из формы
//    args - [необязательный] любые данные, которые будут переданы диалогу
function openLayerDialog(url, title, returnFunc, args) {
    if (!dialogLayer.visible) {
        // Начальная установка свойств модального диалога.
        dialogLayer.url = url;
        dialogLayer.title = title;
        dialogLayer.returnFunc = returnFunc;
        dialogLayer.args = args;
        dialogLayer.returnedValue = "";

        // Загрузка содержимого
        document.getElementById("contentFrame").src = url;

        // Устанавливаем заголовок "окна"
        document.getElementById("barTitle").firstChild.nodeValue = title;

        // Центруем "окно" в окне браузера или фрейме
        dialogLayer.layer.style.visibility = "hidden";
        dialogLayer.layer.style.display = "block"
        centerOnWindow("pseudoWindow");

        // Показываем его и устанавливаем флаг видимости
        dialogLayer.layer.style.visibility = "visible"
        dialogLayer.visible = true;




function closeLayerDialog() {
    dialogLayer.layer.style.display = "none"
    dialogLayer.visible = false;
}
// ************************
// КОНЕЦ КОДА ДИАЛОГА
```

Выполнение библиотеки layerDialog.js начинается с подключения к странице таблицы стилей, соответствующей операционной системе пользователя. Функция getCurrOSUI() определяет тип операционной системы и выбирает один из пяти поддерживаемых стилей. Затем эта функция динамически вписывает в документ тег <link>, содержащий URL нужной таблицы стилей. Код всех пяти файлов .css приведен дальше в листингах с 6.4 по 6.8.

Далее библиотека создает глобальную переменную, работающую как абстрактный объект, хранящий в себе различную информацию о псевдоокне. Вспомогательная функция centerOnWindow(), сделанная на основе рецепта 13.7, центрует псевдоокно при вызове функции, подготавливающей его к отображению. В этом рецепте из функции убран завершающий вызов show(), потому что видимостью окна управляет другая функция библиотеки.

Сразу после загрузки страницы выполняется простая процедура инициализации, загружающая нужный рисунок для закрывающей кнопки. Также она присваивает значение глобальной переменной dialogLayer, содержащей ссылку на нужный слой, для упрощения обращений к нему из других функций.

Основная функция библиотеки, openLayerDialog(), вызывается сценарием, когда ему необходимо отобразить псевдоокно. У этой функции много общих аргументов с функцией для создания модального диалогового окна из рецепта 6.9. Одно из отличий — в новую функцию не нужно передавать размеры окна, поскольку псевдоокно имеет фиксированный размер. Закрывает окно последняя функция библиотеки — doseLayerDialog().

Возможно, вас заинтересует, почему в коде функции openLayerDialog() для того, чтобы отобразить слой, изменяются значения как свойства syle.display, так и свойства style.visibility. Такой код служит для обхода ошибки в IE 6 для Windows. Из-за нее при использовании свойства style.visibility для скрытия окна с экрана прямоугольная область, занимаемая элементом iframe, оставалась пустой (белой). Использование свойства style.display позволяет обойти эту ошибку. Но при этом возникает другая проблема с позиционированием окна перед отображением. Для решения этой проблемы, перед тем как поместить окно по центру, видимость слоя устанавливается в значение hidden, а свойству display присваивается значение block. После этих странных действий можно безопасно использовать свойство visibility для того, чтобы показать пользователю окно.

Псевдоокно не может самостоятельно извлечь строку заголовка из страницы, загруженной в iframe (из-за ограничений, связанных с безопасностью, подобные действия не допускаются, если содержимое псевдоокна загружено с другого сервера или домена). Поэтому текст для строки заголовка необходимо передавать в качестве параметра функции openLayerDialog(). Третьим параметром является ссылка на функцию, вызываемую при щелчке пользователем на кнопке OK или Применить, точно так же, как это делалось в имитированном диалоговом окне из рецепта 6.9. Эта ссылка будет доступна как одно из свойств глобального объекта dialogLayer. То же касается и четвертого параметра функции openLayerDialogO, который можно использовать для передачи дополнительных данных в псевдоокно. Тем не менее следует учитывать, что сценарий в элементе iframe не сможет обратиться к объекту dialogLayer, если страница, на которой он находится, загружена с другого сервера. Для того чтобы отобразить псевдоокно, не передавая ему

никаких дополнительных параметров или ссылок на функции, можно применить следующую команду:

```
openLayerDialog("prefs.html" "Настройки", null, null);
```

В листинге 6.3 показан код библиотеки **LayerDialogDrag.js,** обеспечивающей возможность перетаскивания диалогового окна мышью за элемент в заголовке.

Листинг 6.3. Библиотека layerDialogDrag.js

```
// Глобальная переменная, содержащая ссылку на выделенный элемент
var selectedObj;
// Координаты щелчка мышью относительно элемента
var offsetX, offsetY;

// Установка глобальной ссылки на элемент, который будет перетаскиваться
function setSelectedElem(evt) {
    var target = (evt.target) ? evt.target : evt.srcElement;
    var divID = (target.id== "titlebar") ? "pseudoWindow" : ''
    if (divID) {
        if (document.layers) {
            selectedObj = document.layers[divID];
        } else if (document.all) {
            selectedObj = document.all(divID);
        } else if (document.getElementById) {
            selectedObj = document.getElementById(divID);
        }
        setZIndex(selectedObj, 100);
        return;
    }
    selectedObj = null;
    return;
}
// "Включаем" выбранный элемент
function engage(evt) {
    evt = (evt) ? evt : event;
    setSelectedElem(evt);
    if (selectedObj) {
        if (document.body && document.body.setCapture) {
            // используем перехват событий в IE/Win
            document.body.setCapture();
        }
        if (evt.pageX) {
            offsetX = evt.pageX   ((typeof selectedObj.offsetLeft !=
                "undefined") ?
                    selectedObj.offsetLeft . selectedObj.left);
            offsetY = evt.pageY - ((selectedObj.offsetTop) ?
                    selectedObj.offsetTop . selectedObj.top);
        } else if (typeof evt.clientX != "undefined") {
            offsetX = evt.clientX - ((selectedObj.offsetLeft) ?
                    selectedObj.offsetLeft : 0);
            offsetY = evt.clientY - ((selectedObj.offsetTop) ?
                    selectedObj.offsetTop . 0);
        } else if (evt.offsetX || evt/offsetY) {
            offsetX = evt.offsetX - ((evt.offsetX < -2) ?
                    0 : document.body.scroll Left);
            offsetX -= (document.body.parentElement &&
                    document.body.parentElement.scrollLeft) ?
```

продолжение ↛

Листинг 6.3 (продолжение)

```
                document.body pa rentElement.scroll Left 0
        offsetY = evt.offsetY   ((evt.offsetY < -2) ?
                 0 : document.body.scrollTop);
        offsetY -= (document.body.parentElement &&
                document.body.parentElement.scrollTop) ?
                document.body.parentElement.scrollTop : 0
    }
    evt.cancel Bubble = true:
    return false:
}
// Перетаскиваем элемент
function dragIt(evt) {
    evt = (evt) ? evt : event:
    if (selectedObj) {
        if (evt.pageX) {
            shiftTo(selectedObj, (evt.pageX - offsetX), (evt.pageY
                offsetY)):
        } else if (evt.clientX || evt.clientY) {
            shiftTo(selectedObj, (evt.clientX - offsetX), (evt.clientY
                offsetY)):
        >
        evt.cancel Bubble = true:
        return false:
    }
}
// "Выключаем" выбранный элемент
function release(evt) {
    if (selectedDbj) {
        setZIndex(selectedObj, 0):
        if (document.body && document.body.releaseCapture) {
            // прекращаем перехват событий в IE/Win
            document.body.releaseCapture();
        >
        selectedObj = null:
    }
}
function blockEvents(evt) {
    evt = (evt) ? evt : event:
    evt.cancelBubble = true:
    return false:
}

// Установка обработчиков событий, используемых в Navigator и в IE
function initDragO {
    if (document.layers) {
        // разрешаем перехват этих событий в модели событий NN4
        document.captureEvents(Event.MOUSEDDWN | Event.MOUSEMOVE j
                               Event.MDUSEUP);
        return;
    } else if (document.body & document.body.addEventListener) {
        // разрешаем перехват этих событий в модели событий W3C DOM
       document.addEventListener("mousedown", engage, true):
       document.addEventListener("mousemove", dragIt, true):
       document.addEventListener("mouseup", release, true):
```

```
        return;
    }
    document.onmousedown = engage;
    document.onmousemove = dragIt;
    document.onmouseup = release;
    return;
}
```

Показанная здесь библиотека LayerDialogDrag.js идентична библиотеке из рецепта 13.11, за одним небольшим исключением. Функция setSelectedElement(), которая должна обращаться только к слою titlebar, выбирает в качестве подвижного слоя не его, а внешний слой pseudoWindow. Таким образом, когда пользователь перетаскивает строку заголовка, двигается все окно. Остальная часть библиотеки работает точно так же, как и в рецепте 13.11, включая обработчики событий.

В некоторых ситуациях внешний вид окон может вас расстроить. В тех случаях, когда в содержимом псевдоокна имеются формы, браузеры на медленных компьютерах не могут достаточно быстро обновлять экран, что приводит к временным графическим артефактам при перетаскивании окна. Нужно решить, допустимо ли такое поведение.

Вернемся к таблицам стилей и их влиянию на элементы псевдоокна. При применении таблицы стилей к простейшему HTML-коду элемента div можно получить несколько вариантов оформления, которые показаны на рис. 6.1.

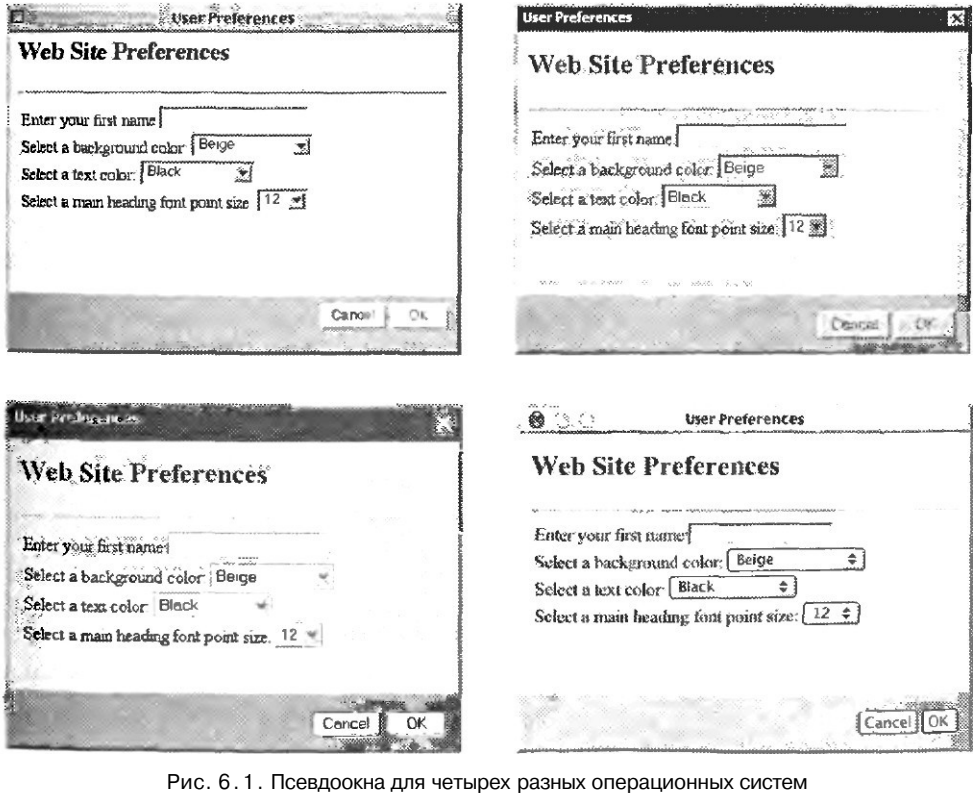


Рис. 6.1. Псевдоокна для четырех разных операционных систем

Одной из основных целей, поставленных при разработке этого приложения, являлось требование как можно большего сходства псевдоокон с обычными окнами операционной системы пользователя. Из-за радикальных изменений, сделанных в интерфейсе Windows XP и Mac OS X, это становится не такой простой задачей. Показанные на рис. 6.1 окна сделаны по образу и подобию окон четырех разных операционных систем. Для того чтобы создать нужный внешний вид, в примере используется пять разных таблиц стилей, управляющих рисунками и размерами элементов. Естественно, должны быть доступны рисунки для заднего фона заголовка окна (нужен только маленький вертикальный кусочек, так как рисунок повторяется) и рисунки закрывающих кнопок. Для того чтобы уловить тонкую разницу между стилями, начнем с описания таблицы стилей dialogLayer_win9xQ.css, предназначенной для IE версий 5, 5.5 и IE 6, работающего в необычном режиме.

Листинг 6.4. dialogLayer_win9xQ.css, **стили для совместимых версий IE для Windows**

```
#pseudoWindow {position:absolute;
               top:0px;
               left:0px;
               width:600px;
               height:502px;
               border:2px solid black;
               background-color:#ffffff;
               border-top:3px solid #cccccc;
               border-left:3px solid #cccccc;
               border-right:3px solid #666666;
               border-bottom:3px solid #666666;
               display:none
              }

#titlebar {position:absolute;
           top:0px;
           left:0px;
           height:16px;
           width:596px;
           background-image:url(titlebar_win9x.jpg);
           color:#ffffff;
           border-bottom:2px solid #666666;
           font-family:Tahoma;
           font-size:8pt;
           font-weight:bold;
           padding:2px;
           text-align:left
          }

#closebox {position:absolute;
           right:0px;
           top:1px
          }

#barTitle {padding-left:3px}
```

```
#contentFrame{position:absolute;
              top:19px;
              left:0px;
              height:477px;
              width:594px;
              background-color:#ffffff;
              margin-left:0px;
              margin-top:0px;
              overflow:visible
             >
```

От версии таблицы стилей, предназначенной для Win9x, эту таблицу отличают лишь небольшие изменения. Сама таблица для Win9x, dialogLayer_win9x.css, показана в листинге 6.5. Эти отличия необходимы для учета различных методов подсчета размера элементов, когда в дело вовлечены границы, рамки и заполнение. Рисунки в этой версии те же, что и в предыдущей, как и сам внешний вид окон.

Листинг 6.5. dialogLayer_win9x.css, таблица стилей для Win9x

```
#pseudoWindow {position:absolute;
               top:0px;
               left:0px;
               width:300px;
               height:102px;
               border:2px solid black;
               background-color:#ffffff;
               border-top:3px solid #cccccc;
               border-left:3px solid #cccccc;
               border-right:3px solid #666666;
               border-bottom:3px solid #666666;
               display:none
              }

#titlebar {position:absolute;
           top:0px;
           left:0px;
           height:16px;
           width:596px;
           background-image:url(titlebar_win9x.jpg);
           color:#ffffff;
           border-bottom:2px solid #666666;
           font-family:Tahoma;
           font-size:8pt;
           font-weight:bold;
           padding:2px;
           text-align:left
          }

#closebox {position:absolute;
           right:0px;
           top:lpx
```

продолжение

Листинг 6.5 (продолжение)

```
#barTitle {padding-left:3px}

#contentFrame{position:absolute;
              top:22px;
              left:0px;
              height:480px;
              width:600px;
              background-color:#ffffff;
              margin-left:0px;
              margin-top:0px;
              overflow:visible
             }
```

В таблице стилей для Win XP, dialogLayer_winXP.css, которая показанна в листинге 6.6, строка заголовка сделана более широкой.

Листинг 6.6. dialogLayer_winXP.css, таблица стилей для браузеров под Win XP

```
#pseudoWindow {position:absolute;
               top:0px;
               left:0px;
               width:600px;
               height:502px;
               border:2px solid black;
               background-color:#ffffff;
               border-top:3px solid #cccccc;
               border-left:3px solid #cccccc;
               border-right:3px solid #666666;
               border-bottom:3px solid #666666;
               display:none


#titlebar {position:absolute;
           top:0px;
           left:0px;
           height:26px;
           width:596px;
           background-image:url(titlebar_winxp.jpg);
           color:#ffffff;
           border-bottom:2px solid #666666;
           font-family:Tahoma;
           font-size:10pt;
           font-weight:bold;
           padding:2px;
           text-align:left
          }

#closebox {position.-absolute;
           right:lpx;
           top:lpx
          }

#barTitle {padding-left:3px}
```

```
#contentFrame{position:absolute:
                top:30px:
                left:0px;
                height:472px;
               width:600px;
                background-color:#ffffff;
                margin-left:0px;
                margin-top:0px;
                overflow: visible
              }
```

В версии для Macintosh, Mac OS X, закрывающая кнопка помещена в левой части заголовка, также сделаны незначительные изменения размера (листинг 6.7).

Листинг 6.7. dialogLayer_macos9.css, стили для Mac OS 9 и более ранних версий

```
   #pseudoWindow {position:absolute;
               top:0px:
               left:0px:
               width:600px:
               height:502px:
               border:2px solid black:
               background-color:#ffffff;
               border-top:3px solid #cccccc:
               border-left:3px solid #cccccc:
               border-right:3px solid #666666:
               border-bottom:3px solid #666666:
               display:none
              }

#titlebar {position:absolute;
           top:0px;
           left:0px;
           height:16px;
           width:596px;
           background-image:url(titlebar_macos9.jpg);
           color:#000000;
           border-bottom:2px solid #666666;
           font-family:Charcoal;
           font-size:9pt:
           font-weight:normal;
           padding:2px;
           text-align:center
          >

#closebox {position:absolute:
           left:0px;
           top:lpx:
           padding-right:3px
          }

#barTitle {padding right:6px;
           background-color:#cccccc;
           padding-left:6px
```

продолжение

Листинг 6.7 (продолжение)

```
#contentFrame{position:absolute;
               top:22px;
               left:0px;
               height:480px;
               width:600px;
               background-color:#ffffff;
               margin-left:0px;
               margin-top:0px;
               overflow:visible
              }
```

Версия для Mac OS X, dialogLayer_macosX.css, сделана с собственными настройками размеров (листинг 6.8).

Листинг 6.8. dialogLayer_macosX.css, версия для Macintosh Mac OS X

```
#pseudoWindow {position:absolute;
               top:0px;
               left:0px;
               width:600px;
               height:502px;
               border:2px solid black;
               background-color:#ffffff,
               border-top:3px solid #cccccc;
               border-left:3px solid #cccccc;
               border-right:3px solid #666666;
               border-bottom:3px solid #666666;
               display:none
              }

#titlebar {position:absolute;
           top:0px;
           left:0px;
           height:16px;
           width:596px;
           background-image:url(titlebar_macosX.jpg);
           color:#000000;
           border-bottom:2px solid #666666;
           font-family:Charcoal;
           font-size:9pt;
           font-weight:normal;
           padding:2px;
           text-align:center
          }

#closebox {position:absolute;
           left:0px;
           top:0px;
           padding-left:5px


#barTitle {padding-right:6px
           background-color:transparent
           padding-left:6px
          }
```

```
#contentFrame{position:absolute;
              top:22px;
              left:0px;
              height:480px;
              width:600px;
              background-color:#ffffff;
              margin-left:0px;
              margin-top:0px;
              overflow:visible
             }
```

Пожалуй, одно из главных достоинств использования при создании псевдоокон позиционируемых элементов заключается в том, что не вызывается никаких дополнительных всплывающих окон. Всплывающие окна не только могут блокироваться различными специальными программами, предотвращающими выполнение window.open(), они также легко могут затеряться среди других окон, приводя пользователей в замешательство. Другое, менее заметное преимущество в том, что в стандарте W3C DOM Level 2 окно почти не выделяется как объект. Спецификации XHTML рекомендуют не использовать фреймы или окна (например, в строгом стандарте XHTML 1.0 атрибут форм и гиперссылок target недействителен). Удержание всего содержимого в одном окне привлекательно для дизайнеров, придерживающихся таких стандартов.

В то же время использование слоев для имитации окон приводит к недостаткам, намного более серьезным, чем замечание о перетаскивании, упомянутое выше. Прежде всего, слой всегда заключен внутри пределов окна, которому он принадлежит. В отличие от нормального окна пользователь не может перетащить псевдоокно так, чтобы оно выходило за границы окна или фрейма. Если содержимое основного окна можно прокручивать, псевдоокно будет прокручиваться вместе с ним.

Хотя в приведенном выше примере псевдоокно имеет фиксированный размер, можно расширить предложенный код, чтобы функция openLayerDialog() получала еще два параметра, определяющие размер окна. Правда, это сложнее сделать, если необходимо поддерживать тот же набор операционных систем, который поддерживает код данного примера. Если посмотреть на установки размеров элементов div, имеющиеся в таблицах стилей, можно заметить, что приходится вычислять большое количество настроек для разных интерфейсов. Например, значения свойств height, width и top для элемента iframe имеют различные значения в разных версиях настроек для Win9x. Если выяснить все факторы, влияющие на размер элементов, и учесть их при формировании окна, можно сделать более гибкую версию, чем показанная в примере. Идеальное применение этого сценария — размещение его в локальной сети, где набор требующих поддержки браузеров и операционных систем весьма ограничен (возможно, даже одной версией!).

Другое применение псевдоокон — создание эквивалента модальных окон, когда блокируется срабатывание ссылок и элементов форм на лежащем ниже окне. Такого эффекта можно достичь, обернув элемент div preudoWindow в другой элемент div, на задний план которого помещено прозрачное изображение. Хитрость в том, чтобы устанавливать размер внешнего элемента в соответствии с размера-

ми основного документа так, чтобы пользователь не смог с помощью полос прокрутки убрать из поля зрения весь документ. После того как это сделано, необходимо поместить псевдоокно в пределах этого элемента, учитывая возможность прокрутки. Во внешнем элементе div должны блокироваться все события. Б него же должны быть помещены обработчики события, используемые при перетаскивании (вместо того, чтобы помещать их в основной документ). Чем в большем числе браузеров нужна поддержка подобной функциональности, тем больше потребуется усилий. Но все это реализуемо, если вы достаточно настойчивы и терпеливы.

Смотрите также

В рецепте 6.8 описывается специфичный для IE способ создания модальных и немодальных диалоговых окон. Рецепт 6.9 содержит описание способа имитации модального окна, применимого для разных браузеров. В рецепте 11.5 показано, как подключать к странице разные таблицы стилей для разных ОС. Более детальное описание библиотеки DHTML2 API — в рецепте 13.3. Способ размещения элемента по центру окна описан в рецепте 13.7. Создание перетаскиваемых мышью элементов показывается в рецепте 13.11. Рецепт 14.14 демонстрирует, как можно динамически переписывать часть содержимого страницы.

7 Управление фреймами

7.0. Вступление

Использование нескольких фреймов, как и использование нескольких окон, вызывает споры среди опытных веб-дизайнеров. Одни их любят, другие не используют вовсе. Некоторые вспоминают то время, когда не все браузеры поддерживали фреймы. Некоторые ветераны веб-дизайна сохраняют это предубеждение до сих пор. Тем не менее в этом есть определенный смысл и сейчас. Строгие реализации стандарта XHTML исключают фреймы из набора команд разметки документа. Формы и гиперссылки, соответствующие этому стандарту, даже не могут иметь атрибут target, с помощью которого можно загрузить документ или результат отсылки формы в другой фрейм.

Однако концепция фреймов не предана забвению. Спецификация XHTML включает в себя версию, использующую фреймы, и в будущем работа W3C должна привести к появлению нового, основанного на XML стандарта команд разметки фреймов (сейчас называемого XFrames). В то же время практически любой современный графический браузер из используемых в настоящее время поддерживает фреймы и будет их поддерживать еще долго. Пользователи же, если установить атрибут border элемента frameset в нулевое значение, могут вообще не узнать, что документ использует фреймы.

Фреймы удобны в некоторых специфических случаях. Обычно их применяют для того, чтобы разбить страницу на две части: большой фрейм, содержащий основную информацию, и фрейм меньшего размера, обычно содержащий оглавление или меню для навигации по сайту. Когда пользователь прокручивает содержимое большего фрейма, меньший остается неподвижным, что упрощает навигацию. Кроме того, навигационный фрейм остается неизменным, тогда как фрейм с информацией отображает разные страницы. За счет этого пользователь избавляется от необходимости перезагружать оглавление при каждом переходе. Еще одно преимущество, которое дает такая стабильность, — это возможность использовать документ, содержащий структуру фреймов или один из фреймов как хранилище данных JavaScript, которые должны сохраняться при переходах между страницами.

Фреймы как окна

С самых ранних дней существования браузеров, поддерживающих сценарии, доступ последних к фреймам осуществлялся через объектную модель браузера. Но не всегда поддержка сценариев соответствует последним стандартам W3C DOM. Изначально каждый фрейм рассматривался как окно (это справедливо и для современных браузеров). Как-никак, фрейм, как и окно, является контейнером для документа. Поэтому большинство свойств и методов, имеющихся у объекта window, применимы при обращении из сценария к фрейму.

Для визуального отображения отношений между фреймами можно использовать понятие иерархии фреймов. Исходный документ, который загружается в браузер и содержит элемент frameset, а также все оформление структуры фреймов, является родительским элементом для фреймов, из которых составлен документ. Как сам родительский фрейм, так и отдельные его дочерние фреймы рассматриваются как объекты window.

Сценарий, выполняющийся в документе, задающем структуру фреймов, может обратиться к каждому из фреймов, а тем самым и к его содержимому через свойство окна frames. Свойство frames представляет собой коллекцию, содержащую ссылки на фреймы, принадлежащие данному окну:

```
window.frames[i]
window.frames["имяФрейма"]
window.имяФрейма
```

Два варианта из предложенных выше полагаются на атрибут фрейма name, который в этом случае должен быть задан. Что же касается числового индекса, то он отсчитывается с нуля, а порядок фреймов определяется порядком их объявления в коде, даже если в определяющем структуру фреймов документе есть много уровней вложения. Другими словами, имеется только один родительский документ, вне зависимости от того, сколько элементов frameset или frame задано в документе.

В том случае, когда документ, на который ссылается атрибут src одного из фреймов, сам представляет собой набор фреймов, ситуация усложняется. Сценарий из самого верхнего набора фреймов может обращаться к подобным вложенным фреймам через следующую иерархию:

```
window.frames["вложенныйнаборФреймов"].frames["имяФрейма"]
```

Сценарий в дочернем фрейме может обращаться как к прочим фреймам своего уровня, так и к основному документу. Свойство parent обеспечивает доступ к документу верхнего уровня, задающему структуру фреймов. Например, обращение к глобальной переменной allLoaded, объявленной в основном документе выглядит так:

```
parent.allLoadeed
```

Для того чтобы обратиться к одному из соседних фреймов, ссылка должна включать в себе обращение к родительскому набору фреймов, из которого мож-

но обратиться ко всем остальным фреймам. В качестве примера рассмотрим следующий простой набор фреймов:

```
<frameset cols="09. *">
    <frame name="navigation" src="navbar.html">
    <frame name="content" src="frameHome.html">
</frameset>
```

Сценарий во фрейме, служащем для навигации по сайту **(navigation),** может обратиться к фрейму с информацией **(content),** используя ссылки такого вида:

```
parent.frames[l]
pa rent.frames["content"]
parent.content
```

Так, сценарий в навигационном фрейме может прокрутить содержимое основного в начало страницы так:

```
parent.frames["content"].scrollTo(0.0);
```

Если документ, загруженный в основной фрейм, сам, в свою очередь, состоит из фреймов, ссылка может стать длиннее, включая в себя путь через последовательность вложений:

```
parent.frames["content"].frames["main"].serollTo(0.0);
```

Для упрощения взаимных ссылок между фреймами при наличии сложной иерархической структуры можно всегда начинать со ссылки на набор фреймов самого верхнего уровня, последовательно спускаясь от нее к нужному документу. Для получения такой ссылки служит свойство top:

```
top frames["content"].frames["main"].scrollTo(0.0);
```

Сценарий из глубоко вложенного фрейма может обратиться к фрейму навигации по сайту, лежащему на верхнем уровне, используя ссылку такого вида:

```
top.frames["navigation"]
```

Пожалуй, важнее, чем все эти способы обращения, знать, что, имея ссылку на фрейм, сценарий имеет немедленный доступ к документу, содержащемуся в этом фрейме. Например, для того чтобы прочитать значение текстового поля в одном из соседних фреймов, можно использовать следующий синтаксис, соответствующий стандартам JavaScript и DOM Level 0:

```
var val = parent.frames["content"].document.entryForm.entryField.value;
```

Нужно только не забыть указать ссылку на документ после ссылки на сам фрейм (это частая ошибка).

Фреймы и наборы фреймов как элементы

В противоположность модели фреймы-как-окна, объектные модели Internet Explorer и W3C позволяют сценариям обращаться к элементам, формирующим фреймы и наборы фреймов. Такие объекты, по существу, не имеют ничего общего

с окнами и во многом аналогичны элементам. Так, эти объекты предоставляют доступ к свойствам, отражающим значения атрибутов тега, например, свойства cols и rows набора фреймов или свойства самого фрейма src и noResize. Такие свойства удобны, когда необходимо считывать или изменять значения атрибутов. Например, в рецептах 7.8 и 7.9 демонстрируется, как можно с помощью сценария настраивать размер фреймов и даже изменять саму структуру фреймов документа.

Возникает вопрос, как сценарий, имея доступ к элементу frame, может обратиться к документу, содержащемуся у него внутри. Решение дает свойство объекта frame, а точнее — одно из двух свойств, зависящих от используемой объектной модели. В объектной модели IE у элемента frame имеется свойство contentWindow, через которое можно обратиться к документу. В стандарте W3C DOM описано свойство contentDocument, содержащее ссылку на объект document внутри фрейма. В Netscape 7 поддерживаются оба этих свойства, но если сценарий должен поддерживать IE 4 и Netscape 6, для получения ссылки на документ можно использовать следующую вспомогательную функцию:

```
function getFrameDoc(frameElem) {
    var doc = (frameElem.contentDocument) ? frameElem.contentDocument
        ((frameElem.contentWindow) ? frameElem.contentWindow.document  null):
    return doc:
}
```

Фреймы и события

Новички, пишущие сценарии, работающие с несколькими фреймами, часто сталкиваются с одной общей проблемой, всегда полагаясь на то, что другой фрейм загружен и нормально работает. Но если один из фреймов документа, быстро загрузившись, обратится к соседнему, который еще не загружен, произойдет ошибка выполнения сценария. Чтобы избежать подобной ситуации, необходимо внимательно использовать обработчики событий onload для самих фреймов и их наборов.

Подобно окну, каждый фрейм имеет событие onload. Это событие происходит, когда все содержимое документа в данном фрейме полностью загружено. Элемент frameset также получает, в свою очередь, событие onload, но только после того, как загружены все составляющие его фреймы (и после того, как в каждом из них произошло событие onload). Таким образом, единственный надежный способ гарантировать нормальную работу функций, обращающихся к различным фреймам, — использовать обработчик события onload набора фреймов. Этот обработчик можно назначить набору фреймов в теге <frameset>.

Следует отличать описанное выше поведение от механизма передачи событий, используемого в объектных моделях IE и DOM. Обработчик события onload для набора фреймов срабатывает только один раз, когда загружается сам набор фреймов и все его дочерние фреймы. Если же пользователь или сценарий изменят URL одного из фреймов, в нем произойдет событие onload, но в наборе фреймов новых событий не произойдет. В IE 5.5 для Windows и в Netscape 7 можно задать обработчик события onload в теге <frame> внутри документа, опре-

деляющего структуру фреймов, который будет срабатывать при каждом изменении адреса страницы, содержащейся в соответствующем фрейме. В более ранних версиях обработчик события onload необходимо встраивать в сам загружаемый документ.

Если документы в наборе фреймов часто меняются и при этом сильно зависят от выполнения сценариев в других фреймах, можно использовать следующий, менее изящный, но эффективный метод проверки доступности содержимого каждого из фреймов. Пусть все документы доставляются с одного и того же домена и сервера (чтобы отвечать требованиям политики безопасности общего источника). Тогда обработчик события в каждом документе может устанавливать глобальный булевский флаг в основном документе. Эта переменная в начале сценария должна инициализироваться значением false, а обработчик события onload устанавливает ее значение в true:

```
<script type="text/javascript">
var loaded = false;

</script>

<body onload="loaded = true">
```

Теперь каждый сценарий, которому необходимо получить доступ к содержимому или сценариям этого документа, может проверять значение переменной loaded, и если оно не равно true, проверять его каждую секунду до тех пор, пока количество попыток не достигнет некоторого предела:

```
// Счетчик числа попыток обращения к другому фрейму
var tries = 0;
// функция, которой необходима информация из другого фрейма
function someFunc() {
    if (parent.otherFrameName.loaded) {
        // OK, другой фрейм готов, используем его в этой ветке кода
        tries = 0;    // подготовка к следующему обращению

    } else if (tries < 5) {
        tries++;
        // повтор через 1 секунду
        setTimeout("someFunc()", 1000);
    } else {
        tries = 0;
        alert("Задача не может быть выполнена.");
```

Скрытые фреймы

Порой разработчики сайтов используют фреймы для поддержания соединения с сервером, чтобы изменения данных на нем немедленно отображались на странице. Технология привязки данных позволяет добиться данного эффекта без особых проблем, но это решение подходит только для продуктов Microsoft (с некоторыми дополнительными ограничениями, касающимися IE для Macintosh). Переносимое решение должно использовать другие методики.

Одно из решений, работающее во многих основных браузерах, основывается на применении невидимою апплета, поддерживающего канал связи с сервером. Двусторонняя связь между апплетом и сценарием обеспечивает свободный обмен данными между двумя средами (в первой реализации этой возможности от Netscape дублировался «LiveConnect»). Тем не менее для успешной работы такого механизма необходимо, чтобы браузер пользователя включал в себя Java runtime environment.

Альтернативой этому решению является создание скрытого фрейма, документ в котором периодически запрашивает с сервера обновленные данных. В этом случае пользователь вообще не сможет узнать о том, что невидимый фрейм обновляется каждые несколько секунд. Он лишь может заметить необъяснимую передачу по сети данных. URL, с которого загружается содержимое невидимого фрейма, может указывать на CGI-программу на сервере. Эта программа должна формировать страницу, в которой нет никакого отображаемого содержимого, а есть только данные с сервера в форме XML или JavaScript (в рецептах 14.4 и 14.5 описываются идеи такого представления данных). Сценарий, дополняющий возвращаемые данные, может передавать данные из невидимого фрейма в ячейки таблиц или узлы HTML на видимой странице. Такой сценарий должен запускаться при выполнении события **onload.**

Чего нельзя делать

Точно так же, как применение всплывающих окон может привести незащищенного пользователя на некоторые неразборчивые в средствах сайты, фреймы демонстрируют сходный набор «дыр». Но, как и в случае с окнами, эти уязвимые места блокированы. Поскольку фреймы оказывают не такое большое влияние на браузер, как окна, список операций, невозможных для фреймов, короче. Даже имея благие намерения, невозможно сделать следующие вещи.

- **Получить доступ к свойствам документов из фреймов, доставленных с другого сервера или домена.** Было бы довольно неприятно, если бы сценарий из одного фрейма мог бы «заглядывать» в другой и извлекать из него адреса или другую информацию. Здесь браузеры применяют так называемую политику безопасности общего источника. Это значит, что сценарий из одного фрейма (или окна) может обратиться к критичным данным другого фрейма (URL, структуры DOM, значения в формах, текст документа) только в том случае, если обе эти страницы доставлены с одного сервера и домена, а также по одному и тому же протоколу.
- **Изменение содержимого поля адреса.** URL, показанный в поле адреса, относится к документу, описывающему структуру фреймов. Поэтому нельзя поместить в эту строку, скажем, адрес содержимого основного фрейма.
- **Формирование в Избранном записей, строго соответствующих структуре фреймов.** В то время как пользователь перемещается по сайту, использующему фреймы, средства браузера по созданию закладок могут обеспечить сохранение либо только адреса самого набора фреймов, либо адрес документа, содержащегося в одном из фреймов. Последнее обычно возможно через контекстное меню браузера. В рецепте 7.6 даются советы по реконструкции структуры фреймов по закладке в одном из фреймов.

7.1. Формирование пустого фрейма в новом наборе

NN2 IE3

Задача

Необходимо, чтобы определение структуры фреймов содержало пустой фрейм, но без необходимости создавать на сервере пустой документ HTML.

Решение

Ниже показан код страницы, определяющей структуру фреймов, в которой один из двух фреймов генерируется с помощью сценария:

```
<html>
<head>
<script type="text/javascript">
<!-
function blankFrame() {
    return "<html><body></body></html>";
}

</script>
</head>
<frameset rows="50, *">
    <frame name="frame1" id="frame1" src="navSlice.html">
    <frame name="frame2" id="frame2" src="javascript:parent.blankFrame()">
</frameset>
</html>
```

Здесь в атрибут src элемента frame помещен псевдо-URL вида javascript:.

Обсуждение

Как вы, наверное, догадались, JavaScript можно использовать для формирования любого HTML-кода, изначально заполняющего фрейм. Например, если в пустом фрейме нужно использовать другой цвет фона, чтобы он совпадал по расцветке с другими фреймами, можно включить в генерируемый тег <body> атрибут bgcolor. Для этого функцию blankFrameO нужно переписать так:

```
function blankFrameO {
    return "<html><body bgcolor="#ccee99"></body></html>"
```

Использование протокола javascript: в атрибуте src — достаточно неоднозначное решение. С одной стороны, это единственный совместимый способ сформировать содержимое элемента, которое обычно поступает из отдельного файла или CGI-процесса на сервере. Но использование URL такого типа приводит к ошибке в том случае, если пользователь отключил JavaScript в своем браузере. Большинство браузеров с отключенным JavaScript в этом случае оставят соот-

ветствующий фрейм пустым, но это может привести к нестабильной работе браузера. Таким образом, применять подобную методику допустимо только в том случае, если вы уверены, что в пользовательских браузерах включен JavaScript.

Следует обратить внимание на то, что использованная ссылка на функцию содержит обращение к родительскому фрейму. Это необходимо, потому что выполнение псевдо-URL javascript: происходит в контексте соответствующего фрейма. А с точки зрения фрейма нужная функция расположена в родительском документе.

Отступая от темы, заметим, что частой ошибкой начинающих разработчиков сценариев является использование javascript: в обработчиках событий. Неизвестно, как возникла эта тенденция, но это избыточно, неправильно и ведет к проблемам. Применять протокол javascript: нужно только там, где требуется URL. Так, его можно задействовать в атрибутах src и href. Использовать его в обработчиках событий нельзя. Точка.

Смотрите также

В рецепте 7.2 показано, как изменить содержимое фрейма после загрузки. Рецепт 14.1 демонстрирует формирование динамического содержимого во время загрузки страницы.

'.2. Изменение содержимого фрейма

NN2 IE3

Задача

Необходимо, чтобы сценарий из одного фрейма изменил документ в другом.

Решение

Изменить URL страницы в соседнем фрейме можно, присвоив новое значение его свойству location.href:

```
parent.имяФрейма.location.href = "newPage.html";
```

Здесь имяФрейма — это имя, присвоенное атрибуту name в теге <frame>, соответствующем нужному фрейму.

Обсуждение

Для того чтобы загрузить нужную страницу во фрейм при щелчке мышью на ссылке, JavaScript не нужен. Необходимо только поместить имя фрейма в атрибут гиперссылки target, а в атрибут href — URL нужной страницы:

```
<a href="products.html" target="content">Каталог продуктов</a>
```

С другой стороны, если вы хотите, чтобы ваши страницы полностью соответствовали стандарту XHTML, атрибут target в гиперссылках использовать нельзя

(документ, задающий структуру фреймов, также не удовлетворяет требованиям XHTML, но он может соответствовать версии XHTML 1.0 для фреймов).

Если необходим атрибут **target,** но использовать его нельзя из-за требований совместимости со стандартами, можно задействовать сценарий, чтобы заполнить значение этого атрибута после того, как документ загружен. Для этого у страницы в качестве обработчика события **onload** можно установить функцию, которая поместит во все ссылки на странице указание на нужный фрейм. Благодаря коллекции document.links, имеющейся в каждой HTML-странице, из сценария можно легко перебрать все гиперссылки и присвоить им нужные значения. Можно также разделить все ссылки на две группы: те, которые должны быть загружены в другой фрейм, и те, которые заместят текущий. Для разбиения элементов на классы можно применить атрибут **class.** Ниже приведена функция (работающая в IE 4, Netscape 6 и старше), которая устанавливает целевой фрейм для всех ссылок, у которых значение атрибута **class** равно **"other":**

```
function setLinksTargets() {
    for (var i = 0; i < document.links.length; i++) {
        if (document.links[i].className == "other") {
            document.links[i].target = "content";
        }
    }
}
```

Помните, что элементы **area,** образующие клиентские карты ссылок, тоже перечислены среди элементов коллекции document.links. Вам решать, включать их в обработку или нет. Для этого достаточно назначить им в HTML-коде нужный класс.

Смотрите также

Рецепт 7.3 показывает, как изменить несколько фреймов в одном действии. Рецепт 7.4 демонстрирует замену текущего набора фреймов полностью новым документом или набором фреймов.

7.3. Изменение нескольких фреймов за один шаг

NN2 **IE3**

Задача

Необходимо, чтобы одна кнопка или гиперссылка меняла содержимое нескольких фреймов сразу.

Решение

Для того чтобы изменить содержимое более чем одного фрейма, понадобится помощь сценария. Наилучший способ — создать отдельную функцию, ответст-

венную за навигацию, и вызывать ее при щелчке на гиперссылке или кнопке. Для обоих элементов управления подойдет примерно такая функция:

```
function loadFrames(url1, url2) {
    parent.otherFrameName.location.href = url1;
    location.href = url2;
    return false;
}
```

Гиперссылка, вызывающая эту функцию, должна выглядеть так:

```
<a href="content12.html" target="content" onclick="return
loadFrames('content12.html', 'navbar12.html')">...</a>
```

Для тех, у кого сценарии отключены, работает ссылка по умолчанию.

Обсуждение

Существует множество вариаций сценария из этого решения. Выбирать одну из них можно, основываясь на желаемом дизайне сайта и том ожидаемом наборе браузеров, которыми будут пользоваться посетители.

В рецепте 2.10 показано, как создать дизайн сайта, при котором поведение браузеров без поддержки сценариев будет определяться стандартными атрибутами гиперссылок href и target. При наличии поддержки сценариев, для того чтобы изменить принятое по умолчанию поведение ссылок, используются обработчики события onclick. Можно пойти по этому пути дальше, сделав ссылки, которые по умолчанию ведут на одну часть сайта, а связанный с ними сценарий обеспечивает переход на другую часть, предназначенную специально для браузеров, поддерживающих сценарии.

Показанный в решении пример рассчитан на использование ссылки, которая меняет документ одновременно как в текущем фрейме, так и в другом. Но нет необходимости использовать только такую комбинацию. Например, можно сделать так, чтобы исходный документ оставался неизменным (например, если это меню навигации по сайту), а несколько других фреймов изменялись при щелчке на ссылке. Для этого следует изменить код функции loadFrame(), подставив в него ссылки на нужные фреймы в том порядке, в котором адреса прибывают из обработчика события. При использовании такого элемента пользовательского интерфейса, как гиперссылки, необходимо удостовериться, что обработчик события onclick возвращает значение false. Это необходимо, чтобы предотвратить поведение гиперссылки, принятое по умолчанию.

Если важна поддержка браузеров, не использующих сценарии, и при этом имеется сложная структура фреймов, можно предложить замену предложенному решению, но при применении сценариев эта задача решается быстрее. Для того чтобы в браузере, не поддерживающем фреймы, можно было менять несколько фреймов одним щелчком на ссылке, следует для каждой комбинации фреймов, которую предлагает меню навигации, сформировать отдельный документ с набором фреймов. При этом в атрибут гиперссылки href необходимо поместить адрес соответствующей ей структуры фреймов, а атрибут target должен

содержать имя специального фрейма верхнего уровня ("_top"). В обработчике события **onclick** продолжайте использовать функцию **loadFrames():**

```
<a href="frameset12.html" target="_top" onclick="return
loadFrames('content12.html'  'navbar12.html')">. .</a>
```

При этом браузер, поддерживающий сценарии, имеет преимущество, поскольку при его применении фреймы будут оставаться на прежних местах, и браузеру не придется лишний раз сравнивать данные из кэша с данными на сервере. Таким образом, при использовании сценариев навигация по сайту будет происходить быстрее, но при этом посетители, не использующие такие браузеры, могут перемещаться по сайту точно так же. Это отличный пример того, как применение **DHTML** может улучшить веб-страницу.

Смотрите также

Изменение содержимого соседнего фрейма демонстрируется в рецепте 7.2.

7.4. Замена набора фреймов страницей

NN2 **IE3**

Задача

Необходимо, чтобы сценарий в одном из фреймов заменил набор фреймов одной страницей, например, для того, чтобы обеспечить переход на другой сайт.

Решение

Для получения такого результата следует назначить нужный URL самому верхнему фрейму в иерархии:

```
top.location.href = "newpage.html"
```

Этот код является аналогом такого описания гиперссылки:

```
<a href="newPage.html" target="_top">...</a>
```

Поэтому сценарий в данном случае следует использовать в том случае, когда замену содержимого подготавливают некоторые другие действия сценария, или если нужно, чтобы страница строго соответствовала спецификации **XHTML,** которая запрещает применение атрибута **target**

Обсуждение

При выборе простых наборов фреймов вместо обращения к **top** можно обращаться к **parent.** В этом случае эти два свойства ссылаются на один и тот же объект. Но обращение к свойству **top** гарантирует обход любых промежуточных наборов фреймов, что может быть важно, если сценарий может выполняться во

вложенном наборе фреймов, в том числе в наборе фреймов, доставленном с другого сервера или домена.

Смотрите также

В рецепте 7.5 показано, как применить эту технику, чтобы предотвратить попадание ваших страниц во фрейм другого сайта.

7.5. Защита от попадания во фрейм другого сайта

NN2 **IE3**

Задача

Необходимо предотвратить ситуацию, когда ваш сайт попадает в набор фреймов чужого сайта.

Решение

В самом начале сценария на странице в головной секции включите следующий код:

```
if (top != self) {
    top.location.href = location.href;
}
```

Если вы применяете фреймы, данный фрагмент сценария необходимо включать только в документе, задающем структуру фреймов. В документах, которые должны загружаться во фреймы, использовать этот код нельзя.

Обсуждение

Ситуации, когда один сайт помещает другой в один из своих фреймов, сейчас не столь распространены, как это было несколько лет назад, но все еще встречаются. Иногда это происходит достаточно невинно, например, когда другой сайт содержит ссылку на ваш, при этом все ссылки на сайте загружаются в отдельный фрейм. Беспокоиться ли о том, что ваш сайт появляется во фрейме другого сайта — ваше личное дело. В то же время корпоративные сайты обычно нуждаются в более полном контроле зрительных впечатлений пользователя, и чей-то баннер, появляющийся в том же окне браузера, весьма нежелателен. К тому же, если сайт находится в чужом наборе фреймов, маловероятно, что пользователь сможет корректно поставить на него закладку. Не все пользователи, особенно некоторые случайные, знают о возможности поставить закладку на страницу (занести ее в Избранное) через контекстное меню браузера.

Сценарий, приведенный в этом рецепте, сравнивает ссылку на текущий объект window со ссылкой, которая указывает на верхнее окно в иерархии фреймов.

К объекту window можно обратиться четырьмя способами: через **window, self, parent** и **top. В** принципе, нет разницы между свойствами **window** и **self,** которые всегда содержат ссылку на текущее окно, независимо от отношений между фреймами и их наборами. Но иногда, как, например, и в этом случае, применение ключевого слова **self** более информативно при чтении кода сценария. Что касается свойства top, то тогда, когда окно браузера не содержит наборов фреймов, оно содержит ссылку на текущее окно. Если при загрузке страницы, содержащей предложенный в этом рецепте сценарий, это равенство не выполняется, набор фреймов замещается текущей страницей.

Так как неизвестный внешний фрейм почти наверняка принадлежит сайту с другого сервера или домена, узнать детали о содержимом окна верхнего уровня невозможно, например, нельзя узнать URL его содержимого. Этому препятствуют ограничения, налагаемые системой безопасности. Но сравнение ссылок на страницы не нарушает политики безопасности.

Смотрите также

В рецепте 7.6 показано, как можно гарантировать, что URL, ссылающийся на один из документов, будет загружен в нужном наборе фреймов.

7.6. Восстановление структуры фреймов

NN2 IE3

Задача

Необходимо, чтобы страница, на которую была поставлена закладка из фрейма, была загружена в нужном окружении.

Решение

Если набор фреймов всегда состоит из одних и тех же документов, для восстановления структуры фреймов можно использовать следующий шаблон кода, который следует включить в начало каждого документа во фрейме. Предлагаемое решение содержит проверку одной особенности Netscape 4, которая описана в секции обсуждения.

```
var isNav4 = (navigator.appName =="Netscape" && parseInt(navigator.appVersion) == 4);
if (top.location.href == self.location.href) {
    if (isNav4) {
        if (window.innerWidth != 0) {
            top.location.href = "frameset12.html";
        }
    } else {
        top.location.href = "frameset12.html";
    }
```

Если пользователь сделал закладку на одну из страниц, то при переходе по ней начнет загружаться соответствующая страница, но затем она загрузит необходимую структуру фреймов.

Обсуждение

Замечание относительно Navigator 4 касается, в частности, версии для Windows. Если в этом браузере пользователь печатает фрейм, его содержимое временно загружается в невидимое окно нулевой ширины. Так как при этом также выполняются сценарии, следует обходить процедуру загрузки фреймов, иначе браузер попадет в бесконечный цикл. Если же среди ожидаемых посетителей не будет пользователей Netscape 4, можно сократить код до такого:

```
if (top.location.href ==self.location.href) {
    top.location.href = "frameset12 html";
}
```

Этот сценарий сравнивает URL текущего окна и окна верхнего уровня. Совпадение этих значений означает, что документ загружается в окно в одиночку, при этом сценарий загружает нужный набор фреймов.

Для случая более гибкого набора фреймов, когда содержимое некоторых фреймов может меняться, следует сделать сценарии и документы более «умными». Сценарий, лежащий в основе каждой страницы, должен не только загрузить набор фреймов, но и передать в этот набор адреса страниц, содержащихся в отдельных фреймах. В качестве примера рассмотрим набор фреймов, в котором имеется статичный фрейм, содержащий каталог продуктов, и основной фрейм, содержимое которого меняется при выборе отдельных пунктов меню. Необходимо, чтобы по закладке на один из документов, содержащих информацию о продукте, загрузился набор фреймов, содержащий и адресуемую страницу.

Для того чтобы такая интеллектуальная загрузка фреймов работала правильно, необходимы сценарии в каждой из страниц, на которые можно поставить закладку, и еще один сценарий в наборе фреймов. Каждая страница передает в страницу с набором фреймов информацию о себе в виде строки поиска, присоединяемой к URL основной страницы. Сценарий в наборе фреймов принимает эту информацию и загружает соответствующие документы.

Начнем рассмотрение с документа, формирующего содержимое. Этот сценарий получает всю необходимую информацию из самой страницы, поэтому его можно оформить в виде внешней библиотеки и подключать к страницам. Вот библиотечный код, выполняемый каждый раз при загрузке страницы:

```
var isNav4 - (navigator.appName == "Netscape" && parseInt(navigator appVersion) == 4):
if (parent ==window) {
    // Не выполняем никаких действий. если NN4 печатает содержимое фрейма
    if (!isNav4 || (isNav4 && window.innerWidth != 0)) {
        if (location.replace) {
            // Если возможно. используем replaceO. чтобы текущая страница не
            // попала в историю посещений
            location.replace("masterFrameset.html?content=" + escape(location.href));
```

```
        } else {
            location.href = "masterFrameset.html?content=" + escape(location.href);
        }
    }
}
```

URL текущей страницы передается в набор фреймов в виде пары имя/значение точно так же, как и строка поиска при отсылке формы. Метод **location.replace()** большинством браузеров, кроме некоторых старых версий, выполняется так, что текущая страница не остается в истории посещений. Если бы эта страница оставалась в истории, то пользователь, щелкнув на кнопке **Назад,** мог попасть в бесконечный цикл, так как браузер не мог бы перейти на предыдущую страницу.

Код в наборе фреймов состоит из двух функций. Одна из них запускается в обработчике события onload тега **<frameset>.** Ключевая вспомогательная функция преобразует строку поиска из URL в объект, имена свойств которого совпадают с именами параметров. Данная функция учитывает возможность передачи нескольких пар имя/значение на тот случай, если вы захотите передать в набор фреймов дополнительную информацию:

```
function getSearchData() {
    var results = new Object();
    if (location.search.substr) {
        var input = unescape(location.search.substr(1));
        if (input) {
            var srchArray = input.split("&");
            var tempArray = new Array();
            for (var i = 0; i < srchArray.length; i++) {
                tempArray = srchArray[i].split("=");
                results[tempArray[0]] = tempArray[1];
            }
        }
    }
    return results;
}
```

У набора фреймов обработчик события **onload** вызывает функцию **loadFrame(),** которая извлекает данные из строки поиска и устанавливает нужный URL для фреима с содержимым:

```
function loadFrame() {
    if (location.search) {
        var srchArray = getSearchData();
        if (srchArray["content"]) {
            self.content.location.href = srchArray["content"];
        }
    }
}
```

Эта функция ищет в переданных данных параметр с именем **content,** таким же, как и имя фрейма. Теоретически можно было бы обобщить эту функцию так, чтобы она назначала переданные адреса фреймам, имена которых перечислены в параметрах. Тем не менее в данном случае забота о том, в какие фреймы будут загружены передаваемые адреса, переложена на отдельные фреймы. При этом

подробности об имени целевого фрейма остаются в контексте сценария набора фреймов.

Если документ, задающий структуру фреймов, содержит для фрейма с информацией о продукте адрес, используемый по умолчанию (то есть берущийся в том случае, если пользователь сразу приходит на основную страницу), документ с этим адресом остается во фрейме, если никаких параметров в URL передано не было. Если же в строке поиска передан URL, указанная по умолчанию страница появляется во фрейме только на короткое время, пока не будет загружен переданный URL.

Смотрите также

В рецепте 7.5 показан сценарий, предотвращающий помещение вашей страницы во фрейм другого сайта. Дополнительные примеры передачи данных через URL показаны в рецепте 10.6.

7.7. Определение размеров фрейма

NN6 **IE4(Win)/5(Mac)**

Задача

Необходимо определить точный размер фреймов в пикселах, после того как набор фреймов отображен.

Решение

Так как нередко один из размеров фрейма указывается в процентах или с использованием символа *, действительный размер фрейма неизвестен до тех пор, пока браузер не завершит все вычисления, связанные с его отображением. Сценарий, выполняемый в одном из фреймов, может определить размеры любого соседнего (или своего собственного) фрейма с помощью следующей функции:

```
function getFrameSize(frameID) {
    var result = {height:0, width:0};
    if (document.getElementById) {
        var frame = parent.document.getElementById(frameID);
        if (frame.scrollWidth) {
            result.height = frame.scrollHeight;
            result.width - frame.scrollWidth;
        }
    }
    return result;
}
```

Данная функция получает в качестве единственного параметра идентификатор фрейма и возвращает объект, свойства width и height которого соответствуют размерам фрейма. Размеры определяются точно в IE 4 для Windows и Netscape 7,

а также в более поздних версиях браузеров. IE для Macintosh версий 5.x возвращает правильные значения, если в рассматриваемом фрейме нет полос прокрутки. Если же полосы имеются, возвращаемый размер уменьшен на их ширину. К сожалению, нет способа определить видимость полос прокрутки во фрейме, если его атрибут scroll содержит значение, принятое по умолчанию (auto).

Обсуждение

Ключевым моментом, определяющим работу функции, является получение ссылок на родительский документ (top) и на соответствующий элемент frame. Таким образом, обычно используемая запись parent.имяФрейма или parent.frames [«имяФрейма»] здесь не поможет. Обращаясь к объекту frame напрямую, можно получить от браузера сведения о размере фрейма, не обращаясь к его содержимому.

Эту функцию можно использовать и в сценарии документа, определяющего структуру фреймов, но при этом следует убрать из ее кода обращение к родительскому документу, так как сценарий уже выполняется в нем. Если поместить данную функцию в набор фреймов, к ней все еще можно будет обращаться из дочерних фреймов, используя следующую запись:

```
var frameSize = parent.getFrameSize("myFrame4");
```

То есть родительский фрейм все равно входит в уравнение тем или иным образом.

Смотрите также

В рецепте 7.8 описан способ изменения размера фрейма с помощью сценария.

7.8. Изменение размера фрейма

NN6 IE4

Задача

Необходимо изменить размер одного из фреймов набора, в том числе установить ему нулевую высоту или ширину, чтобы спрятать фрейм.

Решение

Для решения этой задачи можно использовать вариацию сценария, показанного в листинге 7.1 (в него следует добавить идентификатор набора фреймов) Предлагаемый сценарий расположен в основном документе и вызывается из фреймов. Например, описание кнопки в одном из фреймов, вызывающей функцию toggleFrame(), может выглядеть так:

```
<button onclick="parent.toggleFrame()">Спрятать/показать панель навигации</button>
```

Гиперссылки и изображения также можно использовать как программные выключатели для выполнения подобных действий.

Обсуждение

В браузерах, объектная модель которых обеспечивает доступ к элементам всех типов (IE 4, Netscape 6 и позже), можно управлять атрибутами фрейма **cols** и rows через свойства, имеющие то же имя. В листинге 7.1 показан код библиотеки **frameResize.js,** которая изменяет свойство **cols** типичного набора фреймов из двух колонок. Ширина левого фрейма устанавливается в нулевое значение, тем самым он прячется.

Листинг 7.1. Библиотека для изменения размеров фреймов frameResize.js

```
// Глобальная переменная. сохраняющая предыдущие установки
var origCols:

// Изменение размера левого фрейма
function resizeFrameLeft(left) {
    var frameset = document.getElemetnById('masterFrameset");
    origCols = frameset.cols:
    frameset.cols = left + ". *":
}
function restoreFrame() {
    document.getElementById("masterFrameset").cols = origCols;
    origCols = null:
}
function toggleFrame() {
    if (origCols) {
        restoreFrame();
    } else {
        resizeLeftFrame(0);
    }
}
```

Код начинается с объявления глобальной переменной origCols, служащей для сохранения исходного значения свойства **cols,** чтобы впоследствии настройки можно было восстановить. Идущая далее функция **resizeLeft()** предназначена для того, чтобы устанавливать ширину первого фрейма, при этом прежнее значение сохраняется в переменной **origCols.** Для восстановления исходного размера фрейма служит функция restoreFrame(). Обе эти функции применяются в главной функции, **toggleFrame(),** которую необходимо вызывать из элементов пользовательского интерфейса, расположенных в одном из видимых фреймов.

При разработке интерфейса было бы логичным, если бы текст или изображение элемента управления менялось в зависимости от того, виден фреим или нет. Например, если фрейм виден, подпись должна содержать что-нибудь вроде «Спрятать панель навигации». Если же фрейм невидим, текст должен гласить «Показать панель навигации». Это легко реализовать в браузерах IE, но не в Netscape 6 или 7. Вот как это можно сделать в IE:

```
function toggleFrame(elem) {
    if (origCols) {
       restoreFrameO;
       elem.innerHtml = "&lt;&lt;Спрятать панель навигации":
```

```
    } else {
        resizeFrame(0);
        elem.innerHtml =  'Показать панель навигации&gt; &де:'

}
```

Теперь код кнопки должен выглядеть так:

```
<button onclick="parent.toggleFrame(this)">&lt; &lt;Спрятать панель навигации</button>
```

На рис. 7.1 показаны два состояния кнопки.

Причина, по которой Netscape 6 и 7 не позволяют так же просто управлять пользовательским интерфейсом в том, что эти браузеры автоматически перезагружают содержимое фрейма после того, как его размеры изменены (что совершенно излишне). Таким образом, элементы управления получают исходное значение.

При разработке такого улучшения необходимо также обойти одну весьма неприятную ошибку в некоторых версиях IE, из-за которой обработчик события, вызываемый из родительского документа, получал неправильную информацию об элементе, вызвавшем событие. Чтобы обойти ошибку, следует пойти окольным путем, благодаря чему сценарий будет работать во всех поддерживающих синтаксис W3C DOM браузерах.

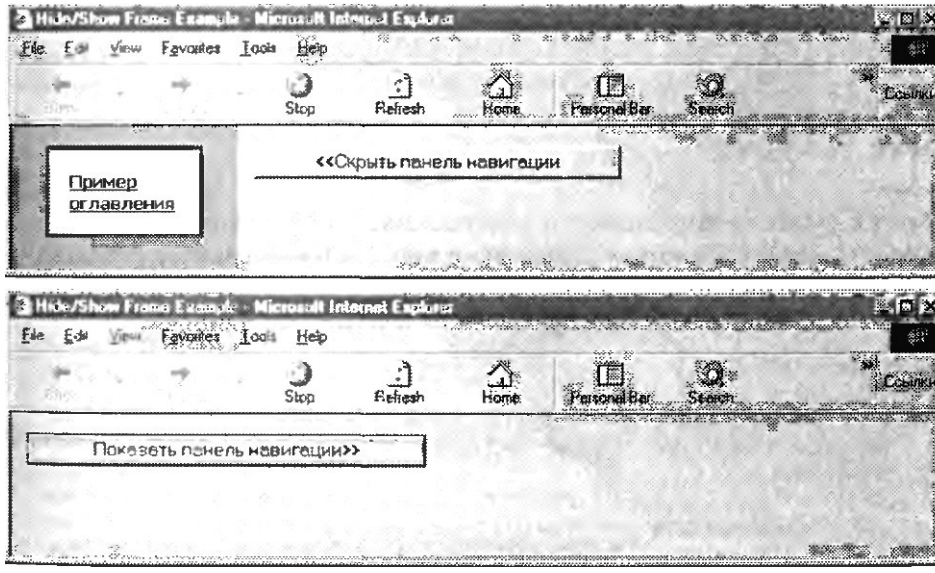


Рис. 7.1. Управление видимостью фрейма

В странице, которая всегда видима, создается пустой элемент span, содержимое которого заполняется, только если браузер имеет необходимую минимальную поддержку сценариев:

```
«span id="togglePlaceholder"></span>
```

Также сценарий использует библиотеку cookies.js (см. рецепт 1.9):

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

Элемент пользовательского интерфейса, используемый для управления видимостью фрейма, формируется сценарием, вызываемым обработчиком события onload:

```
function setToggleUI() {
    var label = "«Hide Navbar";
    if (document.getElementById) {
        if (getCookie("frameHidden") = "true") {
            label = "Show Navbar»":
        }
        var newElem - document.createElement("button");
        newElem.onclick = initiateToggle:
        var newText = document.createTextNode(label);
        newElem.appendChild(newText);
        document.getElementById("togglePlaceholder").appendChild(newElem);
    }
}
```

Обратите внимание на то, что у создаваемого элемента обработчик события onclick вызывает функцию initiateTogle(), объявленную в той же странице. Применение такого обходного пути позволяет передать правильную информации об источнике события в переработанную функцию **toggleFrame()** из родительского документа:

```
function initiateToggle(evt) {
    evt = (evt) ? evt : event:
    var elem = (evt.target) ? evt.target   evt.srcElement;
    if (elem.nodeType = 3) {
        elem - elem.parentNode;
    }
    parent.toggleFrame(elem):
}
```

Теперь функция toggleFrame() в родительском фрейме (который также должен использовать библиотеку cookies.js) не только изменяет ширину фрейма, но и сохраняет текущее состояние документа в cookie и устанавливает новую надпись на кнопке (это нужно потому, что IE не перезагружает страницы).

```
function toggleFrame(elem) {
    if (origCols) {
        elem.firstChild.nodeValue = "«Спрятать панель навигации".
        setCookie("frameHidden", "false". getExpDate(180. 0. 0)):
        showFrame();
    } else {
        elem.firstChild.nodeValue = "Показать панель навигации»":
        setCookieC'frameHidden". "true". getExpDate(180. 0. 0)):
        hideFrame();
    }
}
```

Все остальные функции в родительском документе остаются теми же, что и в исходном решении.

Помните о том, что Netscape 6 и 7 перезагружают содержимое фрейма при изменении его размера, вместо того чтобы просто расширить или сжать его. Мелькание, возникающее от перезагрузки страницы, может раздражать пользователей.

Смотрите также

Определение размера фрейма в пикселах описывается в рецепте 7.7. Библиотека cookies.js подробно описывается в рецепте 1.9.

7.9. Динамическое изменение описания фреймов

NN(нет) **IE 5.5(Win)**

Задача

Необходимо полностью перестроить структуру фреймов с помощью сценария.

Решение

Для решения этой задачи можно применить makeNewFrameset(), одну из двух функций, которая уничтожает текущий набор фреймов и устанавливает новый. При этом сценарии и значения переменных в документе, устанавливающем структуру фреймов, остаются неизменными. Вызвать данную функцию можно из любого элемента управления или из функции в любом из фреймов:

```
parent.makeNewFrameset();
```

Обсуждение

Показанная в листинге 7.2 функция makeNewFrameset() преобразует исходный набор фреймов из двух колонок, доставленный с сервера, в новый набор из двух горизонтальных фреймов. Несмотря на то что многие браузеры должны поддерживать такую функциональность, из-за многочисленных особенностей данный сценарий работает только в IE5.5 для Windows.

Листинг 7.2. Функция для формирования нового набора фреймов в IE5.5

```
function makeNewFrameset() {
    var newFrame1 = document.createElement("frame");
    newFrame1.id = "newFrame1";
    newFrame1.name = "newFrame1";
    newFrame1.src = "altNavBar.html"
    var newFrame2 = document.createElement("frame");
    newFrame2.id = "newFrame2";
    newFrame2.name = "newFrame2";
    newFrame2.src = "altHome.html"

    var frameset = document.getElementById("masterFrameset");
    while (frameset.childNodes.length > 0) {
        frameset.removeChild(frameset.firstChild);
    }
    frameset.cols = null;
    frameset.rows = "80, *";
```

продолжение

Листинг 7.2 (продолжение)

```
    frameset.appendChild(newFrame1);
    frameset.appendChild(newFrame2);
}
```

Прежде чем формировать новый набор фреймов, важно очистить существующие узлы и связанные с ними свойства. В показанной выше функции новые объекты frame формируются сначала в памяти. После того как они готовы, набор фреймов документа очищается от прежних узлов. Таким образом, даже если исходный набор фреймов имел весьма сложную структуру, она исчезает после того, как цикл while выполнит свою работу. Так как в нашем примере структура набора фреймов изменяется с вертикальной на горизонтальную, перед установкой новых значений атрибуту rows исходное значение атрибута cols очищается. В самом конце созданные объекты frame присоединяются к сформированному набору фреймов

Если IE6 — единственный браузер, для которого ведется разработка страницы, можно использовать как временное хранилище для объектов frame объект W3C DOM DocumentFragment. Для этого функцию следует переписать:

```
function makeNewFrameset() {
    var frag = document.createDocumentFragment();
    var newFrame= document.createElement("frame");
    newFrame.id = "newFrame1";
    newFrame.name = "newFrame1".
    newFrame.src - "altNavBar.html"
    frag.appendChild(newFrame);
    newFrame = document.createElement("frame");
    newFrame.id - "newFrame2":
    newFrame.name = "newFrame2";
    newFrame.src = "altHome.html"
    frag.appendChild(newFrame);

    var frameset = document.getElementById("masterFrameset");
    while (frameset.childNodes.length > 0) {
        frameset removeChild(frameset.firstChild);
    }
    frameset.cols = null:
    frameset.rows = "30%, *":
    frameset.appendChild(frag);
}
```

Если же необходима обратная совместимость, всегда можно вернуться к использованию метода document.write() во фрейме, адресуемом свойствами frame или top. При этом все содержимое нового документа, формирующего структуру фреймов, следует поместить в строку. Затем вызовите метод top.document.write(), завершив его вызовом top.document.close(). При этом все следы исходного документа верхнего уровня (включая переменные как в самом наборе фреймов, так и в отдельных фреймах) исчезнут, их заменит новый набор фреймов.

Смотрите также

Изменение размеров фреймов описывается в рецепте 7.8. Использование метода document.write для формирования новой страницы описывается в рецепте 14.2 (там же описываются методы подключения библиотек сценариев к таким страницам).

8 Динамические формы

8.0. Вступление

Придание формам интеллектуальности всегда стимулировало развитие языка JavaScript и объектной модели DOM. В то время как в языке происходили значительные изменения, JavaScript постоянно использовался в формах, чтобы создать мгновенно реагирующий, дружественный интерфейс, разработка которого без использования сценариев потребовала бы передачи данных на сервер и обратно (что приводит к задержкам).

Обращение к формам и элементам управления

До появления стандарта W3C DOM программисты использовали для доступа к формам и их элементам управления синтаксис, который сейчас называется DOM Level 0. При использовании этого старого соглашения полагаются по большей части на идентификаторы, присвоенные элементам управления в форме. Фактически и сейчас браузер не будет отсылать данные формы на сервер, если элементам в ней не присвоены имена (несмотря на то, что сейчас более распространено использование атрибута id). Тем не менее сейчас объектная модель предоставляет доступ к формам через массивы форм и элементов, с помощью которых можно получать нужные значения, применяя индексы. Рассмотрим пример, где в документе имеется единственная форма, названная **userInfo**. Совместимый со старым синтаксисом сценарий должен брать для обращения к форме следующий код:

```
document.forms[0]
document.forms["userInfo"]
document.userInfo
```

В каждом объекте формы имеется массив elements, содержащий ссылки на все элементы управления в форме. Например, если вторым элементом управления в форме **userInfo** является текстовое поле **age**, существует три способа обратиться к этому элементу:

```
document.userInfo.elements[1]
document.userInfo.elements["age"]
document.userInfo.age
```

Обратите внимание на то, как показанный синтаксис отображает иерархию вложения элементов, от документа, через форму к элементу управления. Это позволяет применять одно и то же имя элемента в разных формах на одной странице, что невозможно (или, по крайней мере, не рекомендуется) при работе с атрибутом id.

В браузерах, поддерживающих доступ к элементам через атрибут id, например в IE 4 и NN 6 и более поздних версиях, можно обращаться к формам напрямую, используя синтаксис объектной модели, поддерживаемой браузером. Например, в IE 4 (а также в более поздних версиях IE) можно применять синтаксис Microsoft DOM:

```
document.all.formID
document.all.formControlID
```

При привлечении синтаксиса W3C DOM (поддерживаемого, начиная с версий IE 5 и NN 6) следует использовать стандартный способ обращения к элементам:

```
document.getElementById("formID")
document.getElementById("formControlID")
```

Хотя идентификаторов вполне достаточно для обращения к элементам из сценария, в каждой форме, отправляемой на сервер, необходимо назначить идентификатор не только атрибуту id, но и атрибуту name. При этом можно брать один и тот же идентификатор для обоих атрибутов, не опасаясь конфликта имен.

Таким образом, синтаксис, служащий для обращения к элементам форм, диктуется браузерами, которые необходимо поддерживать. Если вас заботит только обратная совместимость (в том числе и с Navigator 4), стоит остановиться на синтаксисе DOM Level 0, который сейчас поддерживается большинством основных браузеров и будет поддерживаться еще долго.

Стратегии проверки форм

Проверка форм на стороне клиента перед отправкой на сервер — хороший способ ускорить исправление возможных ошибок при заполнении полей. Эта методика не может заменить проверку данных на сервере, но, как и прочие DHTML-технологии, проверка форм на стороне клиента позволяет увеличить эффективность работы пользователя.

Даже если рассматривать только проверку на стороне клиента, есть две основные стратегии: стратегия реального времени и стратегия пакетной проверки. При проверке в реальном времени сценарий отслеживает признаки действий пользователя, такие как события **onchange** в полях ввода, и немедленно проверяет вводимые данные на соответствие наложенным на поле ограничениям. Достоинства такого подхода в том, что пользователь в любой момент видит данные, которые будут отправлены на сервер. Другими словами, лучше сразу указать пользователю, что адрес электронной почты введен неправильно, и вернуть его к нужному полю, чем сообщить об этом позже. С другой стороны, в большинстве случаев нельзя заставлять пользователя заполнять поля в определенном порядке, как это

будет происходить при использовании для проверки значений события onblur. Стоит позволять пользователю перемещаться между полями ввода в процессе заполнения.

Однако при этом следует проверить все данные непосредственно перед отправкой формы. Такая проверка предоставляет последний шанс перепроверить данные перед отправкой. В большинстве случаев для этого пригодны те же процедуры, которые использовались для проверки в реальном времени. Запускать пакетную проверку данных может либо событие формы onsubmit, либо обыкновенный элемент управления наподобие кнопки.

Кроме того, нужно решить, использовать при анализе строк старомодные функции разбора строк или применять современные, основанные на регулярных выражениях средства JavaScript. Регулярные выражения — быстрое и мощное средство для шаблонов в текстовых данных, но их синтаксис довольно сжат и непонятен. Некоторые примеры обоих подходов можно увидеть в рецепте 8.2, а во вступлении к главе 1 представлен краткий обзор регулярных выражений в JavaScript. Однако для полного понимания всего значения регулярных выражений необходим более подробный учебник.

Отправка по почте и возвращаемые страницы

Обычно форма в HTML-документе отсылает свои данные программе, выполняющейся на сервере. Эта программа может быть написана на одном из множества языков: Perl, Java, Visual Basic, C, C++, C#, Tcl и многих других. При этом клиент никак не может узнать или повлиять на то, как данные обрабатываются на сервере. Обычно эти данные принимаются серверной программой, разбиваются на части, и полученная информация сохраняется в базе данных или участвует в других операциях с хранилищами данных на сервере. После того как обработка завершена, сервер возвращает клиенту HTML-страницу, которая по умолчанию отображается в том же окне и фрейме, что и форма. В этом процессе нет ничего необычного.

Однако нередко авторы не имеют доступа к сценариям на сервере либо по причине закрытости хостинга, либо вследствие отсутствия опыта работы с необходимыми технологиями. Для того чтобы заполнить этот пробел и узнать, какие данные отправлены на сервер, можно использовать в атрибуте формы action URL mailto:. Когда такая возможность впервые появилась в браузерах, данные просто молча отправлялись по указанному электронному адресу. Но после того, как Сеть охватила забота о безопасности (особенно с появлением сценариев), незаметная отсылка была упразднена. Ей на смену пришел более явный процесс, когда пользователь с правильно настроенным браузером и почтовой программой может увидеть сообщение перед тем, как оно будет отправлено.

Сложность при использовании такой методики состоит в том, что у многих пользователей почтовая программа не настроена или же их отпугивает вид открывшегося окна почтовой программы. К тому же нет подтверждения, что почта отправлена правильно и целиком. Более того, не существует абсолютно надежного способа сформировать простую страницу подтверждения. Для того чтобы

можно было надежно произвести такое действие, необходимо получить подтверждение от почтовой программы, что можно показать новую страницу взамен формы. Если сценарий заменит страницу до того, как передача данных будет завершена, может быть потеряна большая часть отправленных данных прежде, чем они прибудут на сервер.

Одно из решений — поискать в Сети хостинг для Unix-программы, называемой FromMail. Эта программа позволяет пересылать форму определенной серверной программе, которая может передать данные формы по указанном адресу электронной почты.

8.1. Начальная установка фокуса

NN2 **IE3**

Задача

Необходимо сделать так, чтобы пользователь мог начать вводить данные в форму без необходимости вручную устанавливать фокус на поле ввода.

Решение

Считая, что форма и текстовое поле на странице всегда имеют одно и то же имя, можно использовать событие onload для установки фокуса:

```
<body onload="document.имяФормы.имяПоля.focus()">
```

Вместо имяФормы и имяПоля нужно подставить реальные имена формы и элемента ввода в ней.

Обсуждение

Сколь бы ни было простым это решение, вы будете удивлены, насколько оно может ускорить работу пользователей с формой. Например, форма поиска Google использует подобный механизм, поэтому все, что нужно сделать, — просто выбрать сайт google.com из избранного и начать набирать текст запроса. Без помощи сценария пользователям приходилось бы каждый раз щелкать мышью на текстовом поле или несколько раз нажимать клавишу Tab, чтобы перенести фокус. Фокус на элемент управления нужно устанавливать только после того, как страница полностью загружена, так как в противном случае некоторые браузеры могут переместить его на другой элемент после загрузки.

Смотрите также

В рецепте 8.4 показано, как можно автоматически выбрать то поле, которое не прошло проверку. Автоматический переход между текстовыми полями с фиксированной длиной текста демонстрируется в рецепте 8.10.

8.2. Обычные проверки текста

NN2 IE3

Задача

Необходимо удостовериться, что текстовое поле содержит произвольный текст, число, строку фиксированной длины или адрес электронной почты.

Решение

Для решения данной задачи можно использовать библиотеку функций проверки форм, показанную в секции обсуждения (листинги 8.1 и 8.2). Эта библиотека содержит следующие функции:

`isNotEmpty()`

Проверяет на наличие хотя бы одного символа в поле.

`isNumber()`

Проверяет поле на наличие в нем числа.

isLenl6()

Поле содержит ровно 16 символов.

`isEmailAddr()`

В поле содержится корректный адрес электронной почты.

Для проверки полей формы во время ввода данных можно использовать событие **onchange,** передавая в функцию проверки ссылку на проверяемый элемент через ключевое слово **this.** Например, показанное ниже поле ввода должно использоваться для ввода адреса электронной почты:

```
<input type="text" size="30" name="eMail" id="eMail"
    onchange="if (isNotEmpty(this)) {isEMailAddr(this)}" />
```

Как объединить эти функции для выполнения пакетной проверки перед отсылкой формы, показано в рецепте 8.3. Возвращаемые функциями проверки значения важны для успешной отсылки данных.

Обсуждение

В листинге 8.1 показан набор полностью совместимых функций проверки текста. Все эти функции вызываются как из обработчика события onchange, так и при пакетной проверке данных формы. Всем этим функциям передается ссылка на проверяемый элемент формы.

Листинг 8.1. Совместимые функции проверки текстового поля

```
// функция проверки того, что текстовое поле не пусто
function isNotEmpty(elem) {
    var str = elem.value;
```

продолжение

Листинг 8.1 (продолжение)

```
    if (str o null || str.length =- 0) {
    alert("Пожалуйста заполните поле."):
    return false:
    } else {
    return true:
    }
}

// проверка того. что в поле положительное или отрицательное число
function isNumber(elem) {
    var str = elem.value;
    var oneDecimal - false:
    var oneChar = 0:
    // удостоверяемся. что значение не преобразовано в число
    str = str.toString0:
    for (var i = 0; i < str.length; i++) {
        oneChar = str.charAt(i).charCodeAt(0);
        // первым символом может быть знак минус
        if (oneChar ==45) {
            if (i o 0) {
                continue;
            } else {
                alert("Знак минус может быть только первым" +
                    " символом строки.")-
                return false:
            }
        }
        // десятичная точка
        if (oneChar <= 46) {
            if (!oneDecimal) {
                oneDecimal = true:
                continue:
            } else {
                alert ("В числе может быть только одна десятичная точка!"):
                return false:
            }
        }
        // символы вне диапазона от 0 до 9 не подходят
        if (oneChar < 48 || oneChar > 57) {
            alert("В поле разрешено вводить только числа."):
            return false;
        }
    }
    return true:
}

// проверка того. что текст в поле длиной 16 символов
function isLen16(elem) {
    var str = elem.value;
    if (str.length != 16) {
        alert("Нужно указать 16 символов.");
        return false:
    } else {
        return true:
    }
```

```
// Проверка того, что текст в поле можно интерпретировать как адрес
// электронной почты
function isEMailAddr(elem) {
    var str = elem.value;
    str = str.toLowerCase();
    if (str.indexOf("@") > 1) {
        var addr = str.substring(0, str.indexOf("@"));
        var domain = str.substring(str.indexOf("@") + 1, str.length);
        // требуется как минимум один домен верхнего уровня
        if (domain.indexOf(".") == -1) {
            alert("Неправильно указан домен.");
            return false;
        }
        // сначала символ за символом анализируем первую часть адреса
        for (var i = 0; i < addr.length; i++) {
            oneChar = addr.charAt(i).charCodeAt(0);
            // на первом месте не может стоять точка или тире, точка не
            // может быть последним символом
            if ((i == 0 && (oneChar ==45 || oneChar == 46)) ||
                (i == addr.length - 1 && oneChar ==46)) {
                alert("Неправильно введена первая часть адреса.");
                return false;
            }
            // допустимы символы: (-  _ 0-9 a-z)
            if (oneChar ==45 || oneChar ==46 || oneChar ==95 ||
                (oneChar > 47 && oneChar < 58) ||
                (oneChar > 96 && oneChar < 123)) {
                continue;
            } else {
                alert("Неправильно введена первая часть адреса.");
                return false;
            }
        }
        for (i=0; i < domain.length; i++) {
            oneChar = domain.charAt(i).charCodeAt(0);
            if (i == 0 && (oneChar ==45 || oneChar == 46)) ||
                ((i == domain.length - 1 || i == domain.length - 2) &&
                oneChar ==46)) {
                alert("Неправильно указан домен.");
                return false;
            }
            if (oneChar ==45 || oneChar ==46 || oneChar =-95 ||
                (oneChar > 47 && oneChar < 58) || (oneChar > 96 &&
                oneChar < 123)) {
                continue;
            } else {
                alert("Неправильно указан домен.");
                return false;
            }
        }
        return true;
    }
    alert("Некорректный адрес e-mail.");
    return false-
}
```

В случае сложных проверок (например, проверки адреса электронной почты), использование регулярных выражений делает функции проверок более компактными, но такое решение требует большой аккуратности и многочисленных проверок для того, чтобы убедиться, что все работает так, как нужно. На листинге 8.2 показаны эквивалентные функции проверки, построенные на регулярных выражениях.

Листинг 8.2. Функции проверки текста, построенные на регулярных выражениях

```
// Проверка того, что в текстовом поле есть хотя бы один символ
function isNotEmpty(elem) {
    var str = elem.value;
    var re = /.+/;
    if(!str.match(re)) {
        alert("Пожалуйста, заполните поле.");
        return false;
    } else {
        return true;
    }
}
//Проверка наличия положительного или отрицательного целого числа
function isNumber(elem) {
    var str = elem.value;
        var re = /^[-]?\d*\.?\d*$/;
    str = str.toString();
    if (!str.match(re)) {
        alert("Пожалуйста, введите в это поле число.");
        return false;

    return true;
}
// Проверка того, что в поле ровно 1 символ
function isLen16(elem) {
    var str = elem.value;
    var re = /\b.{16}\b/.
    if (!str.match(re)) {
        alert("необходимо ввести 16 символов.");
        return false;
    } else {
        return true;
    }
}
// Проверка того  что в поле правильный адрес e-mail
function isEMailAddr(elem) {
    var str = elem.value;
    var re = /^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]{2,7}$/;
    if (!str.match(re)) {
        alert("Проверьте формат адреса.");
        return false;
    } else {
        return true;
    }
}
```

Данный набор функций снабжает пользователя менее детальной информацией о том, какого рода произошла ошибка, по сравнению с совместимой версией библиотеки. Предоставить дополнительную информацию возможно, но для этого придется разделить регулярные выражения на части для определения совпадающих частей. В первой функции, isNotEmpty(), регулярное выражение ищет строку, состоящую хотя бы из одного произвольного символа (.+). В функции isNumber(), проверяющей правильность числа, регулярное выражение ищет строку, начинающуюся (^) с нуля или одного знака минус ([-]?), за которым следует ноль или больше цифр (\d*), ноль или одна десятичная точка (\.?) и еще ноль или больше цифр (\d*), завершающих ($) строку. В функции isLenl6() шаблон ищет строки фиксированной длины. При этом с обоих концов строки отыскиваются границы слов (\b), а между ними — ровно 16 произвольных символов (.{16}). Для того чтобы пользователь не ввел больше 16 символов, можно также ограничить максимальную длину текста в поле шестнадцатью символами.

В замысловатом шаблоне, примененном в функции isEailAddr(), ищется строка, которая начинается (^) с одной или нескольких букв, цифр, знаков подчеркивания или дефисов ([\w-]+). После этих символов должен идти знак @, за которым следует одно или несколько имен доменов или компьютеров, разделенных точками (([\w-]+\.)+). Завершать ($) строку должны от двух до семи букв в любом регистре, означающих имя домена верхнего уровня ([a-zA-Z]{2,7}). Под такой шаблон не подходят адреса с явным указание IP-адреса сервера после знака @, но спецификация электронной почты (Internet Engineering Task Force RFC 822) в любом случае не рекомендует такую форму записи.

В дополнение к использованию процедур проверки по отдельности иногда их необходимо каскадировать. Например, ни одна из показанных процедур проверки чисел, строк фиксированной длины и адресов не проверяет, содержится ли в строке вообще что-нибудь. Поэтому, если, например, в форме имеется поле ввода адреса, то в обработчике события onchange можно вызывать сначала функцию isNotEmpty(), а затем isEmailAddr(), причем сделать это так, чтобы при провале первой проверки вторая не выполнялась. Здесь в игру и входят возвращаемые функциями булевские значения:

```
<input type="text" size="30" name="eMail" id="eMail"
    onchange="if (isNotEmpty(this)) {isEMailAddr(this)}" />
```

Среди продемонстрированных процедур не показана одна, проверяющая правильность ввода даты. Дело в том, что проверка правильности ввода даты — весьма непростая операция из-за огромного разнообразия форматов и соглашений, применяемых в мире. Везде, кроме intranet-приложений, в которых все пользователи заведомо будут использовать один и тот же формат даты, лучше применять для ввода дат три отдельных поля (или элемента select): для ввода даты, месяца и года. Можно также задействовать обработчик события формы onsubmit для того, чтобы объединить данные из всех трех полей в одну строку, помещаемую в скрытое поле, из которого сервер может напрямую передавать данные в базу. Для выбора даты также можно взять всплывающий календарь, показанный в рецепте 15.9, который помогает пользователю выбирать дату, оставляя ее форматирование сценарию. Наконец, если все пользователи предпочи-

тают один и тот же формат, можно применить методики проверки дат, описанные в рецепте 2.12.

Смотрите также

В рецепте 8.3 показана структура, производящая пакетную проверку данных формы с использованием только что описанных функций. Автоматическая установка фокуса на элементе, не прошедшем проверку, демонстрируется в рецепте 8.4. Идеи относительно проверки дат можно почерпнуть из рецепта 2.12.

8.3. Проверка перед отсылкой

NN2 **IE3**

Задача

Необходима функция проверки, которая отменила бы передачу формы на сервер в случае некорректных данных, чтобы пользователь мог исправить ошибки.

Решение

Пакетная проверка данных формы обычно запускается из обработчика события формы onsubmit. При этом отсылка данных отменяется, если хотя бы одна функция проверки вернула false. Чтобы добиться такого поведения, необходимо включить в обработчик события оператор return, который будет возвращать булевское значение, полученное от функции проверки:

```
<form .. onsubmit=" return validateForm(this);">
```

Обсуждение

Функцию пакетной проверки формы можно реализовать как основную функцию, вызывающую функции проверки отдельных полей. Для того чтобы это продемонстрировать, рассмотрим простую форму с различными типами элементов управления на ней. Текстовые поля формы будут использовать проверку в реальном времени, а событие формы onsubmit будет вызывать пакетную проверку. Оба типа проверок будут основаны на функциях, показанных в рецепте 8.2 (можно применять как традиционный их вариант, так и основанный на регулярных выражениях). Вот код формы:

```
 <form method="GET" action="cgi-bin/register.pl"
    name="sampleForm" onsubmit="return validateForm(this)">
Имя: <input type="text" size="30" name="name1  id= name1"
    onchange="isNotEmpty(this)" /><br>
Фамилия: <input type="text" size="30" name="name2" id="name2"
    onchange="isNotEmpty(this)" /><br>
Адрес E-mail: <input type="text" size="30" name="eMail" id="eMail"
    onchange="if (isNotEmpty(this)) {isEMailAddr(this)}" /><br>
```

```
Регион: <select name="continent" id="continent">
    <option value="" selected>Выберите страну</option>
    <option value="Africa">Африка</option>
    <option value="Asia">Азия</option>
    <option value="Australia">Австралия</option>
    <option value="Europe">Европа</option>
    <option value="North America">Северная Америка</option>
    <option value="South America">Южная Америка</option>
</select><br>
Лицензионное соглашение:
    <input type="radio" name="accept" id="accept1" value="agree" />Принимаю
    <input type="radio" name="accept" id="accept2" value="refuse" />Не принимаю
<br>
<input type="reset" /> <input type="submit" />
</form>
```

Внешний вид этой формы можно увидеть на рис. 8.1. Обычно, пока пользователь перемещается между полями формы, проверяется только поле ввода электронного адреса. Если же перехо ить от одного поля формы к другому, не делая в них никаких изменений, обработчик события onchange вызываться не будет (что является еще одной причиной необходимости общей проверки формы перед отсылкой). Помимо обычных проверок, в этой форме необходимо удостовериться в том, что пользователь выбрал один из вариантов в элементе select, а также указал свое отношение к лицензионному соглашению с помощью переключателя (возможно, некоторые дизайнеры усомнятся в необходимости делать переключатели без выбранного по умолчанию элемента, но здесь начальный выбор не нужен).

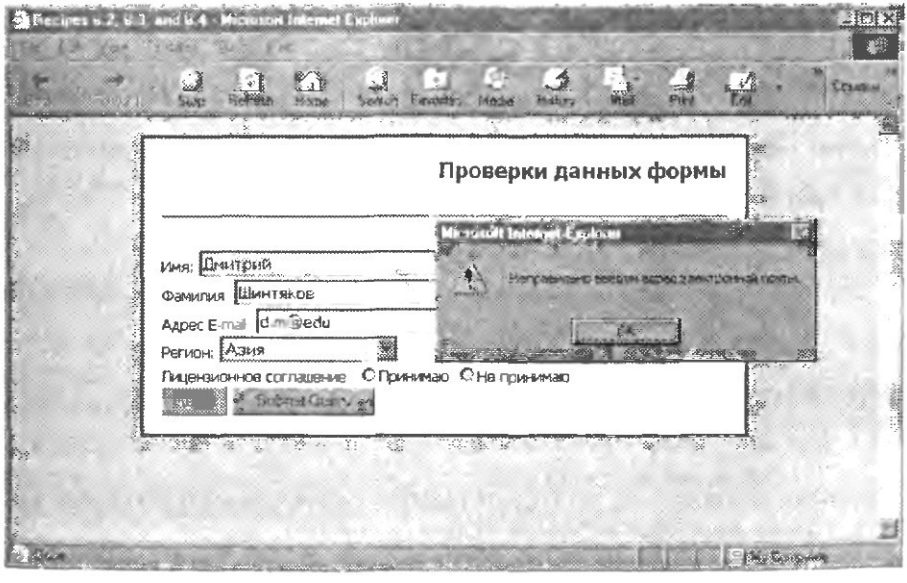


Рис. 8.1. Форма с различными типами элементов

После того как пользователь щелкает на кнопке отправления формы, вызывается функция validateForm(), проверяющая осе элементы формы. Вызовы

проверяющих функций множатся, поэтому если в форме имеется несколько ошибок и пользователь исправил одну из них, функция при повторном щелчке на кнопке отправления сообщает о следующей:

```
function validateForm(form) {
    if (isNotEmpty(form.name1)) {
        if (isNotEmpty(form.name2)) {
            if (isNotEmpty(form.eMail)) {
                if (isEMailAddr(form.eMail)) {
                    if (isChosen(form.continent)) {
                        if (isValidRadio(form.accept)) {
                            return true:
                        }
                    }
                }
            }
        }
    }
    return false:
}
```

Функции для проверки значений элемента select и группы переключателей не описаны в рецепте 8.2. Их код следует ниже:

```
// Проверка того, что пользователь сделал выбор  отличный
// от значения по умолчанию
function isChosen(select) {
    if (select.selectedIndex== 0) {
        alert("Выберите значение из списка."):
        return false:
    } else {
        return true:
    }
}
// Проверка того, что пользователь выбрал один из переключателей
function isValidRadio(radio) {
    var valid = false:
    for (var i = 0: i < radio.length: i++) {
        if (radio[i].checked) {
            return true:
        }
    }
    alert("Выберите одно из значений.");
    return false.
}
```

Обратите внимание на то, что оформление элемента select предполагает, что первую строку из списка выбирать нельзя.

Все отдельные функции проверок связаны с общей управляющей функцией. Если хотя бы одна из проверок не удачна, координирующая функция возвращает значение false, приводя к тому, что обработчик события onsubmit также возвращает false. Это, в свою очередь, отменяет отсылку данных формы. Если же все функции проверок вернули значение true, координирующая функция

также возвращает true, что приводит к нормальному продолжению отправки данных формы.

Полагаясь на обработчик события onsubmit, следует иметь в виду, что пользователь мог отключить выполнение сценариев в своем браузере, чтобы обойти проверку данных на стороне клиента. Поэтому разумно дублировать проверки на сервере. Если вы все же предпочитаете все проверки делать на стороне клиента и известно, что все пользователи будут использовать браузеры с поддержкой сценариев, для отсылки данных формы можно взять обычный элемент ввода типа button вместо настоящей кнопки отсылки типа submit. Пусть обработчик события кнопки onclick теперь вызывает функцию проверки, и в случае если проверка завершилась благополучно, метод объекта формы submit(). В таких условиях пользователь, отключивший сценарии, вообще не сможет отправить никаких данных.

Смотрите также

Отдельные функции проверок описаны в рецепте 8.2. Как выбрать элемент формы, содержащий ошибку, и выделить в нем текст, рассказывается в рецепте 8.4. Некоторые идеи о проверке правильности ввода дат предлагаются в главе 2.12.

8.4. Установка фокуса на неправильное поле

NN2 **IE3**

Задача

Необходимо переместить фокус в незаполненное или заполненное неправильно текстовое поле и выделить в нем весь текст для ускорения исправления ошибки.

Решение

Решение основывается на использовании методов текстового поля focus() и select(). Неприятная ошибка согласования в IE для Windows приводит к тому, что эти функции нельзя вызывать прямо из функции проверки. Для того чтобы дать окну с предупреждением время исчезнуть, а остальной части страницы — стабилизироваться, необходима некоторая задержка перед установкой фокуса.

Вот пример функции, которая устанавливает фокус на поле и выделяет его содержимое:

```
function focusElement(formName, elemName) {
    var elem = document forms[formName] elements[elemName];
    elem.focus();
    elem.select();
```

Теперь следует изменить процедуры проверки текста так, чтобы сразу после закрытия окна с предупреждением, но перед выполнением оператора return вызывалась с помощью setTimeout() показанная выше функция. Вот пример функции проверки формата адреса электронной почты, которая теперь устанавливает фокус на неправильно заполненное поле (добавления выделены жирным шрифтом):

```
function isEMailAddr(elem) {
    var str = elem.value;
    var re = /^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]{2,7}$/;
    if (!str.match(re)) {
        alert("Неправильно введен адрес электронной почты.");
        setTimeout("focusElement('" + elem.form.name + "." + elem.name +
            "')", 0);
        return false;
    } else {
        return true;
    }
}
```

Информацию об именах формы и элемента в ней можно передать в функцию в виде строк благодаря тому, что первый параметр функции setTimeout() имеет строковый тип данных.

Обсуждение

Метод focus() (но не метод select()) можно вызывать и для других типов элементов, например для переключателей в группе или для элементов select Но при этом необходимо удостовериться, что метод вызывается для нужного объекта. Например, в случае группы переключателей атрибут name одинаков для всех переключателей в группе. Можно установить фокус на отдельный переключатель в группе, используя его индекс в массиве элементов (или через уникальный идентификатор, если такая информация имеется у сценария).

Для того чтобы отличать друг от друга различные элементы управления в формах, следует применить свойство type. Для элементов ввода это поле может содержать следующие значения: button, checkbox, file, hidden, image, password, ratio, reset, submit или text (в устаревших браузерах изображения не используются в сценариях). Элементу select может соответствовать значение select-multiple или select-one. Таким образом, можно переписать функцию focusElement() так, чтобы она не вызывала метод select() у тех элементов, которые не предлагают текстового ввода.

Смотрите также

Рецепт 8.2 содержит функции проверки форм, которые можно адаптировать для использования с методикой автоматической установки фокуса. В рецепте 8.2 описана методика пакетной проверки данных формы, которая автоматически вызывает установку фокуса на ошибочных элементах.

8.5. Смена адреса формы

NN2 IE3

Задача

Необходимо отсылать данные формы на разные адреса в зависимости от действий пользователя.

Решение

Наиболее часто встречается случай, когда на форме имеется две или больше кнопок, отправляющих данные на сервер, и при этом каждая кнопка передает данные отдельной CGI-программе на сервере. При использовании элементов submit можно применить для изменения URL серверной программы, получающей данные, обработчик события onclick. Новый URL следует поместить в свойство формы action:

```
<input type="submit" value="Послать в Штаб"
 onclick=this.form.action="http://www.megacorp.com/submitSpesc.asp" />
<input type="submit" value="Послать на оценку"
 onclick=this.form.action="http://www.megacorp.com/reviewComm.asp" />
```

Кроме того, изменить свойство action может любая функция, выполняющаяся до того, как форма отошлет свои данные.

Обсуждение

Следует с осторожностью применять это решение для тех браузеров, среди которых могут быть не поддерживающие сценарии (или же сценарии были в них отключены). Сперва вы можете предпочесть задать URL, на который данные формы будут отсылаться по умолчанию, если сценарии недоступны. Но если, как в этом примере, есть два элемента submit, обе кнопки будут отсылать данные по одному и тому же адресу, что может запутать посетителя, если его браузер не исполняет сценарии.

Конечно, не обязательно применять только элементы input типа submit. Например, в форму можно поместить флажок, состояние которого можно использовать как признак действия, которое необходимо выполнить. В этом случае перед отправкой данных обработчик события onsubmit должен проверять состояние флажка и помещать в свойство action соответствующий URL:

```
form.action = (form.myCheckbox.checked) ? "cgi-bin/special.pl"
    "normal.jsp":
```

Нужно не забывать, что атрибут action в формах обязателен по всем современным правилам проверки HTML и XHTML.

Смотрите также

В рецепте 8.3 рассматривается использование обработчика события onsubmit для обработки данных формы перед отправкой.

8.6 Блокирование отправки при нажатии Enter

NN4 (только) IE4

Задача

Нужно предотвратить отправку формы при нажатии клавиши Enter/Return.

Решение

Хотя это и не описывается стандартами HTML, в большинстве браузеров форма, содержащая только одно текстовое поле, отсылается на сервер при нажатии клавиши Enter/Return, если фокус принадлежит этому полю. Хотя для некоторых форм это удобно, такое поведение может привести к преждевременной отправке данных формы, в которой есть другие требующие внимания элементы.

Чтобы игнорировать нажатие клавиши Enter в текстовом поле, можно использовать обработчик события onkeydown этого поля:

```
function blockEnter(evt) {
    evt = (evt) ? evt : event:
    var charCode = (evt.charCode) ? evt.charCode
        ((evt.which) ? evt.which · evt.keyCode);
    if (charCode ==13) {
        return false:
    } else {
        return true.
    }
}

<input type="text" name="search" size="40"
 onkeydown="return blockEnter(event)" />
```

К сожалению, в Netscape 6 и 7 обработчик события onkeydown функционирует неправильно, так что этот рецепт не работает в этих браузерах.

Обсуждение

Для клавиши Enter (расположенный над правой клавишей Shift) как код символа, так и код клавиши равны 13. На клавиатурах Macintosh имеется отдельная клавиша Enter, которой соответствует символ с кодом 3, ее поведение дублирует Enter в формах. Обычно отсылка данных происходит при приходе события onkeydown, следовательно, это событие и нужно использовать для блокирования нужных клавиш. В поведении IE для Macintosh имеется небольшое отличие. Нажатие клавиши Enter (которая на клавиатурах Macintosh называется Return), всегда вызывает отсылку формы, вне зависимости от того, сколько в ней текстовых полей. Таким образом, для того чтобы воспрепятствовать отсылке данных во всех браузерах, следует установить блокирующий код в обработчики событий у всех текстовых элементов ввода на форме.

Если форма используется просто в качестве контейнера для элементов управления, предназначенных для ввода и отображения данных на стороне клиента (как, например, калькулятор ипотечных платежей), она вообще не должна никуда отсылаться. В этом случае самый простой способ блокировать возможную отсылку данных — замкнуть накоротко обработчик события onsubmit следующим кодом:

```
<form ... onsubmit="return false:">
```

Такой подход можно использовать как один из способов обойти особенности поведения Netscape 6 и 7, из-за которых невозможно использовать обработчик события onkeydown для блокирования отправки. Для этого добавьте в форму кнопку или ссылку, которая будет вызывать метод формы **submit(),** при вызове которого не происходит запуск обработчика события onsubmit. Таким образом, необходимо блокировать отсылку с помощью onsubmit, а для передачи данных на сервер использовать сценарий:

```
<input type="button" value="Готово" onclick="this.form.submit();">
```

Такой подход позволяет позаботиться обо всех нажатиях **Enter** и одновременно дает пользователю возможность отправлять данные. Можно также продолжать использовать пакетный механизм проверки формы, несмотря на то что событие **onsubmit** блокировано. Вместо него функцию проверки формы должен вызывать перед отправкой обработчик события **onclick:**

```
<input type="button" value="Готово"
onclick="if (validateForm(this.form)) {this.form.submit()}">
```

Смотрите также

В рецепте 8.7 показывается, как использовать клавишу **Enter** для перемещения фокуса на следующее поле в последовательности Автоматический перенос фокуса в смежных текстовых полях фиксированной длины демонстрируется в рецепте 8.12.

8.7. Перенос фокуса с помощью Enter

NN4 IE4

Задача

Нужно сделать так, чтобы нажатие клавиши **Enter** в текстовом поле переносило фокус в следующее текстовое поле в форме.

Решение

Каждому полю, которое при нажатии **Enter** должно перемещать фокус в другой элемент, необходимо назначить обработчик события onkeypress. Этот обработчик должен вызывать функцию focusNext(), показанную в обсуждении. При этом

обработчик события должен передать в функцию ссылку на форму и имя следующего элемента в цепочке переходов, а также объект события (в соответствии с моделью событий W3C в Netscape). Например, поле ввода, названое name1, может передавать фокус ввода полю **name2** так:

```
<inout type="text" name="name1" id="name1"
    onkeypress="return focusNext(this.form, 'name2' event)">
```

Обсуждение

Хоть поведение форм HTML отлично от поведения форм в самостоятельных приложениях для работы с базами данных, с помощью сценария можно передвигать фокус от одного поля к другому нажатиями клавиши **Enter.** Написание нужного сценария несколько утомительно, потому что в каждом обработчике события следует указать следующий элемент в цепочке. Для начала необходимо блокировать возможность отослать данные формы, используя в теге **<form>** обработчик **onsubmit="return false;".** Что касается текстовых полей, то каждое из них должно вызывать следующую функцию:

```
function focusNext(form, elemName, evt) {
    evt - (evt) ? evt . event:
    var charCode = (evt.charCode) ? evt.charCode
        ((evt.which) ? evt.which : evt.keyCode):
    if (charCode == 13) {
        form.elements[elemName].focus();
        return false:
    }
    return true:
}
```

Чтобы проверять данные в реальном времени (при вводе), можно продолжать использовать обработчик события onchange.

Смотрите также

Автоматический переход между смежными текстовыми полями, содержащими строки фиксированной длины, демонстрируется в рецепте 8.12.

8.8. Передача данных по нажатию Enter в любом поле

NN4 IE4

Задача

Необходимо, чтобы нажатие **Enter** в любом текстовом поле приводило к отправке данных формы.

Решение

Чтобы имитировать работу самостоятельного приложения для работы с базой данных, можно использовать автоматическую установку фокуса (показанную в рецепте 8.6) во всех элементах, кроме последнего, который использует обработчик onkeypress для вызова следующей функции:

```
function submitViaEnter(evt) {
    evt = (evt) ? evt : event;
    var target = (evt.target) ? evt.target : evt.srcElement;
    var form = target.form;
    var charCode = (evt.charCode) ? evt.charCode
        ((evt.which) ? evt.which : evt.keyCode);
    if (charCode ==13) {
        if (validateForm(form)) {
            form.submit();
            return false;
        }
    }
    return true;
}
```

Здесь мы не рассматриваем проверку данных при вводе. Показанная выше функция производит общую (пакетную) проверку данных формы (при этом в случае ошибки пользователь получает сообщение об этом) и отсылает данные, используя метод submit(). Обработчик события onkeypress у последнего текстового поля в форме должен выглядеть так:

```
<input type="text"      onkeypress="return submitViaEnter(event)">
```

Данная методика рассчитана на то, что обработчик события onsubmit у формы равен «return false», чтобы данные формы можно было бы отослать, используя в сценарии метод submit() (это обсуждается в рецепте 8.6).

Обсуждение

Чтобы помочь обобщить добавления к формам, описанные как в рецептах 8.6 и 8.7, так и в этом рецепте, ниже показан код текстовой формы с обработчиками событий, которая вызывает некоторые из функций, описанные в предыдущих главах:

```
<form action="    " method="GET" name="sampleForm" onsubmit="return false">
Имя: <input type="text" size="30" name="name1" id="name1"
    onkeypress="return focusNext(this.form, 'name2'. event)"
    onchange="isNotEmpty(this)" /><br>
Last Name: <input type="text" size="30" name="name2" id="name2"
    onkeypress="return focusNext(this.form. 'eMail' event)"
    onchange="isNotEmpty(this)" /><br>
Адрес Email: <input type="text" size="30" name="eMail" id="eMail"
    onkeypress="return submitViaEnter(event)" /><br>
<input type="reset" /> <input type="button" value="Готово
    onclick="if (validateForm(this.form)) {this.form.submit( )}" />
</form>
```

Следует обратить внимание на то, что здесь отсылает данные обыкновенная кнопка типа button. Из-за этого не поддерживающие сценарии браузеры не смогут отослать данные такой формы. Если же необходима работа формы с такими браузерами (при этом проверки придется производить на сервере), можно использовать модифицированный элемент submit, как это показано дальше:

```
<input type="submit" onclick="if (valdateForm(this.form))
   {this.form.submit()}" />
```

Благодаря этому те браузеры, которые могут выполнять сценарии, блокируют нормальное функционирование кнопки submit, а браузеры без такой поддержки могут отослать форму как обычно.

Смотрите также

Рецепт 8.6, в котором описывается, как блокировать ненужную отсылку данных формы при нажатии на Enter Использование Enter для перемещения между полями ввода демонстрируется в рецепте 8.7.

8.9. Блокирование элементов формы

NN6 IE4

Задача

Необходимо временно блокировать элемент управления на форме, чтобы пользователь не мог к нему обращаться.

Решение

В браузерах IE 4, NN 6, а также в более поздних версиях у элементов форм есть булевское свойство disabled, которое в любой момент времени может быть изменено из сценария. Чтобы заблокировать элемент формы, можно применить любую действительную ссылку на него:

```
document.myForm.myFirstTextBox. disabled = true;
```

Для восстановления работы элемента нужно установить его свойство disabled в значение false:

```
document.myForm.myFirstTextBox.disabled - false:
```

Блокированный элемент управления обычно отображается серым цветом и не позволяет пользователю менять свое значение.

Обсуждение

На рис. 8.2 показана форма с рис. 8.1, у которой блокированы элемент управления. Значения таких элементов управления не будут отправлены на сервер, несмотря на то что сценарии могут считывать и изменять их значение.

Свойство disabled в IE 4 для Windows доступно у всех отображаемых элементов управления, но это свойство не полностью наследуемо. Например, если у формы установить свойство disabled в значение true, все вложенные в нее элементы (и подписи к ним) будут выглядеть блокированными, но пользователи все равно смогут изменять их значение. Если в IW для Windows необходимо блокировать как элементы управления, так и подписи к ним, следует одновременно блокировать как сами элементы управления, так и содержащую их форму. Блокирование элементов, не принадлежащих формам в IE 5 для Macintosh и в Netscape 6 и 7 не поддерживается.

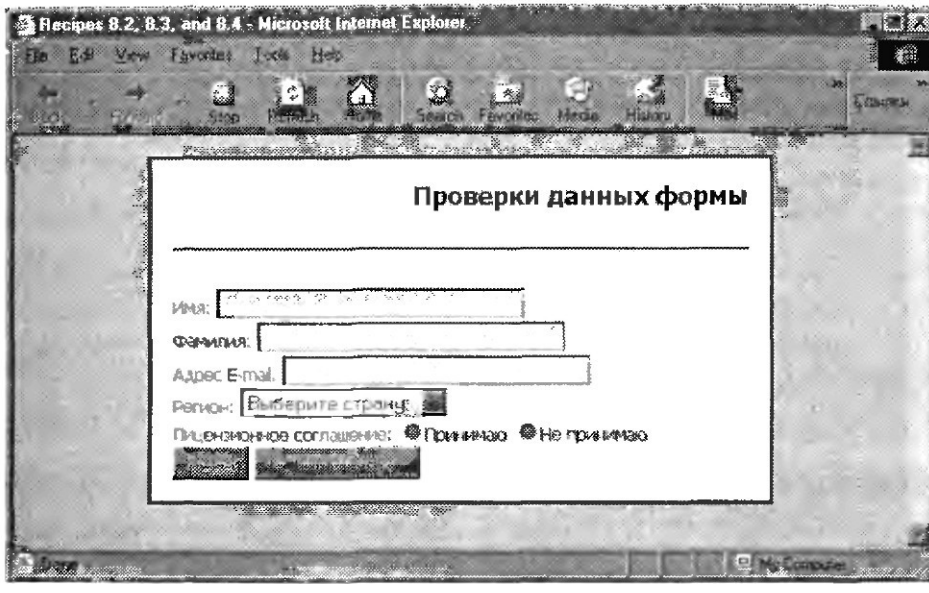


Рис. 8.2. Форма с блокированными элементами управления

Смотрите также

В рецепте 8.10 показывается, как скрыть часть формы до тех пор, пока она не понадобится. Скрытие и отображение элементов демонстрируется в рецепте 12.7.

8.10. Скрытие элементов формы

NN6 **IE4**

Задача

Необходимо держать определенные элементы управления скрытыми до тех пор, пока они не понадобятся в ответ на изменение значения некоторых других элементов.

Решение

Элементы управления, содержащие дополнительные подробности, можно оставить невидимыми до тех пор, пока пользователь не выберет нужное значение в элементе select или не установит некоторый флажок. Для скрытия и отображения группы связанных элементов управления можно использовать функцию, подобную togglePurDec(), описанной в секции обсуждения.

Обсуждение

Скрытие группы дополнительных элементов управления может служить альтернативой их блокированию (см. рецепт 8.9). Например, в следующей выдержке из формы заказа для магазина форма содержит несколько элементов управления, которые остаются невидимыми до тех пор, пока пользователь не ответит «Да» на вопрос номер 3:

```
<form name="survey" ...>

<p>3. Принимаете ли вы решения о покупках в вашей компании?<br>
<input type="radio" id="purDecFlag0" name="purchaseDecision"
onclick="togglePurDec(event)">Нет
<input type="radio" id="purDecFlag1" name="purchaseDecision"
onclick="togglePurDec(event)">Да
<div id="purchaseDecisionData" style="display:none: margin-left:20px">
<p>
3a. Каков бюджет ваших приобретений за текущий налоговый год?
<select name="PurBudget">
<option value="">Выберите одно из значений:</option>
<option value="1">Меньше $50.000< option>
<option value="2">$50.000-100.000</option>
<option value="3">$100.000-500.000</option>
<option value="4">$500.000+</option>
</select>
</p>
<p>
3b. Какую роль вы играете в принятии решения о покупке? (отметьте все. что верно)<br>
<input type="checkbox" name="purRole1">Поиск<br>
<input type="checkbox" name="purRole2">Рекомендации<br>
<input type="checkbox" name="purRole3">Обзор чужих рекомендаций<br>
<input type="checkbox" name="purRole4">Подписывание чужих заказов<br>
<input type="checkbox" name="purRole5">Ничего из вышеперечисленного<br>
</p>
</div>
</p>
<p>4. Как долго вы занимаете текущую должность?
<select name="emplLen">
<option value="">Выберите один·из вариантов:</option>
<option value="1">Меньше 6 месяцев</option>
<option value="2">6-12 месяцев</option>
<option value="3">1-2 лет</option>
<option value="4">2+ лет</option>
```

```
</select>
</p>

</form>
```

На рис. 8.3 показаны два состояния этой формы.

Видимостью необязательного фрагмента формы управляют обработчики события onclick двух переключателей, используя показанную ниже функцию **togglePurDec()**:

```
function togglePurDec(evt) {
    evt = (evt) ? evt : event;
    var target = (evt.target) ? evt.target : evt.srcElement;
    var block = document.getElementById("purchaseDecisionData");
    if (target.id == "purDecFlag1") {
        block.style.display = "block";
    } else {
        block.style.display = "none";
    }
}
```

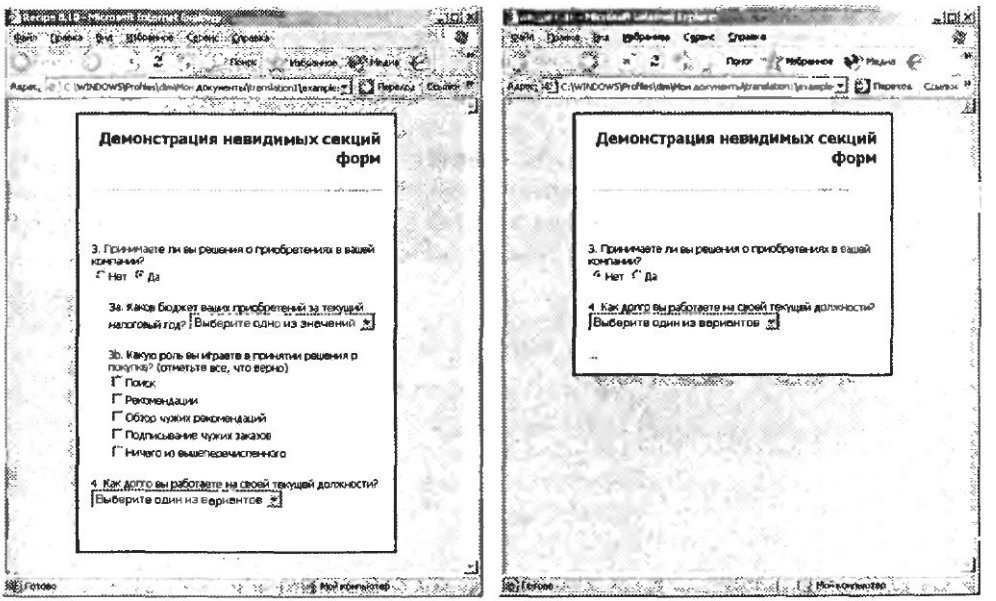


Рис. 8.3. Скрытие дополнительных элементов управления на форме

В Netscape 4 можно имитировать технику скрытия элементов управления, поместив их в слой, у которого свойство **visibility** управляет тем, видим ли блок. Но есть серьезная проблема: в Navigator 4 каждый слой содержит свой собственный документ. Это означает, что одну форму нельзя разбить на несколько слоев. В каждом слое будет отдельная форма, которая содержит элементы управления, соответствующие этому блоку, поэтому сценарии должны воссоединить данные из нескольких отдельных слоев и поместить их в главную форму перед отсылкой. Все это следует делать осторожно, но это возможно. Что касается IE 4,

Netscape 6, а также более поздних версий, то в них последовательность рассматривается как единая форма.

Смотрите также

Блокирование элементов форм демонстрируется в рецепте 8.9. В рецепте 8.12 содержатся дополнительные примеры, использующие скрытие элементов форм.

8.11. Ограничение ввода только цифрами (или буквами)

NN4 **IE4**

Задача

Необходимо ограничить символы, которые можно вводить в текстовое поле, цифрами, буквами или другими символами из фиксированного набора.

Решение

В браузерах IE 4, Netscape 4 и более поздних версиях можно блокировать выполнение события onkeypress в том случае, если нажат неподходящий символ. Показанная ниже функция блокирует нажатия всех клавиш, кроме цифр от 0 до 9, если вызывать ее из обработчика события onkeypress у текстового поля:

```
function numeralsOnly(evt) {
    evt - (evt) ? evt : event;
    var charCode - (evt.charCode) ? evt.charCode r ((evt.keyCode) ? evt.keyCode :
        ((evt.which) ? evt.which : 0));
    if (charCode > 31 && (charCode < 48 || charCode > 57)) {
        alert("В это поле можно вводить только цифры.");
        return false:
    }
    return true:
}
```

Чтобы использовать эту функцию, необходимо настроить обработчик события следующим образом:

```
<input type="text"     onkeypress="return numeralsOnly(event)">
```

Чтобы сделать собственный вариант этой функции, разрешающей ввод нужных вам символов, следует использовать значения **charCode.**

Обсуждение

Рассмотрим логику работы оператора if в показанной выше функции numeralsOnly(). Этот оператор блокирует все символы, коды которых больше чем 31 и которые не являются ASCII-кодами цифр. Важно, чтобы символы с кодами меньше 31

не блокировались. Эти символы не являются буквенно-цифровыми, например, клавиши возврата (8), табуляции (9) и ввода (13). Обычно не нужно блокировать возможность их использования.

Если необходимо, можно объединять несколько диапазонов символов. Например, если помимо цифровых символов следует разрешить пользователю вводить запятую, необходимо добавить еще один блок в выражение так, чтобы оператор блокировал только те символы, которые не удовлетворяют прежним условиям и не являются запятой (ее код 44):

```
if (charCode > 31 && (charCode < 48 || charCode > 57) && charCode !=44)
```

Кроме того, необходимо проверять, не ввел ли пользователь больше, чем одну десятичную запятую.

При создании текстового поля, пропускающего только буквы, следует учитывать различие между буквами в верхнем и нижнем регистрах. Символы в разном регистре имеют разные ASCII-коды, лежащие в разных числовых диапазонах (см. приложение А). Можно выбрать трудоемкий путь, преобразуя коды в символы, которые затем преобразуются в верхний или нижний регистр, однако гораздо проще проверять попадание кода в один из двух диапазонов. Ниже показана функция, которая пропускает латинские буквы в любом регистре (но не знаки пунктуации):

```
function lettersOnly(evt) {
    evt = (evt) ? evt    event:
    var charCode = (evt.charCode) ? evt.charCode · ((evt.keyCode) ? evt.keyCode ·
        ((evt.which) ? evt.which : 0)):
    if (charCode > 31 && (charCode < 65 || charCode > 90) &&
        (charCode < 97 || charCode > 122)) {
        alert("В это поле можно вводить только буквы."):
        return false:
    }
    return true.
}
```

В рецепте 8.9 приведена функция для Internet Explorer, которую можно применять для преобразования символов нижнего регистра в символы верхнего в момент ввода. Если же база данных на сервере требует символов верхнего регистра, помимо такой функции можно использовать процедуру, которая будет приводить регистр символов при проверке данных:

```
form.fiels.value = form.field.value.toUpperCase()
```

Наконец, вот еще один пример функции, позволяющей вводить в поле символы из некоторого ограниченного набора. Предположим, что одна из таблиц базы данных требует того, чтобы в ее поле могла быть помещена строка, состоящая только из одного символа, Y или N (Yes или No). Фильтрующая функция для ввода только таких символов должна выглядеть так:

```
function ynOnly(evt) {
    evt - (evt) ? evt : event:
    var charCode - (evt.charCode) ? evt.charCode : ((evt.keyCode) ?
       evt.keyCode : ((evt.which) ? evt.which : 0)):
    if (charCode > 31 && charCode != 78 &&  charCode != 89 &&
        charCode != 110 && charCode !- 121) {
        alert("Можно вводить только символы \"Y\" или \"N\".");
```

```
        return false;
    }
    return true;
}
```

Кроме того, в этом случае необходимо делать небольшую дополнительную настройку, чтобы в поле нельзя было ввести более одного символа:

```
Подпись: <input type="text" name="signature" size="2" maxlength="1" onkeypress="return
ynOnly(event)" /> (Y/N)
```

Смотрите также

Проверка символа, введенного пользователем до того, как символ достигнет текстового поля, демонстрируется в рецепте 9.8.

8.12. Автоматический переход между полями фиксированной длины

NN4 **IE4**

Задача

Необходимо, чтобы курсор продвигался от одного поля к другому по мере ввода текста.

Решение

В показанном ниже фрагменте формы пользователь должен ввести номер кредитной карточки в четыре поля, каждое из которых вмещает четыре символа.

```
Номер кредитной карты:
<input type="text" name="cc1" size="5" maxlength="4"
    onkeypress="return numeralsOnly(event)"
    onkeyup="autofocus(this, 4, 'cc2', event )">  
<input type="text" name="cc2" size="5" maxlength="4"
    onkeypress="return numeralsOnly(event)"
    onkeyup="autofocus(this, 4, 'cc3', event)">  
<input type="text" name="cc3" size="5" maxlength="4"
    onkeypress="return numeralsOnly(event)"
    onkeyup="autofocus(this, 4, 'cc4', event)">  
<input type="text" name="cc4" size="5" maxlength="4"
    onkeypress="return numeralsOnly(event)">
```

В каждом из полей обработчик события onkeypress блокирует ввод всех символов, кроме цифр (смотрите рецепт 8.11). Кроме того, обработчик события onkeyup для передачи фокуса в следующее поле вызывает показанную ниже функцию после того, как введено нужное количество символов:

```
function autofocus(field, limit, next, evt) {
    evt = (evt) ? evt : event;
```

```
    var charCode - (evt.charCode) ? evt.charCode . ((evt.keyCode) ?
       evt keyCode : ((evt.which) ? evt.which   0)):
    if (charCode > 31 && field.value.length == limit) {
        field.form.elements[next].focus():
    }
}
```

В этом примере последнее поле никуда не передает фокус, но можно, например, передавать его на следующее поле в форме.

Обсуждение

Существует множество вариаций на тему представленной в этом рецепте функции autofocus(). Хотя можно переделать функцию для работы с заранее известным набором полей (при этом имя поля отделяется от его номера, и номер увеличивается: cc1, cc2, cc3 и т. д.), обычно удобнее обобщить ее, чтобы повторно использовать для других наборов полей или в других приложениях. Вторым параметром функции является максимальное количество символов, которое можно вводить в одно поле. Используя этот параметр, можно применить функцию для организации ввода, когда отдельные сегменты имеют разную длину, например серийные номера некоторых программ.

Другая возможность — отказаться от использования второго аргумента, limit, определяя максимальное количество символов в поле из его свойства maxlength. Если поддержка Navigator 4, не предоставляющего такого свойства, не важна, можно исключить второй аргумент и изменить условное выражение в операторе if следующим образом:

```
if (charCode > 31 && field.value.length — field.maxlength) {
```

Первая проверка кода символа в этом операторе необходима для того, чтобы пользователь мог перемещаться между полями в обратном направлении, используя сочетание клавиш Shift+Tab. Если же включить в ограничение на число символов коды нижней части ASCII-таблицы, можно попасть в замкнутый цикл передачи фокуса.

Смотрите также

Как ограничить вводимые в поле символы определенным набором прямо при вводе, показано в рецепте 8.11.

8.13. Замена содержимого элемента select

NN4 IE4

Задача

Нужно изменить варианты, предлагаемые элементом select.

Решение

Для начала необходимо полностью очистить содержимое элемента select:

```
document myForm mySelect.options.length - 0.
```

Затем следует заполнить список новыми объектами option:

```
document.myForm.mySelect options[0] - new Option("Экстра". "xtra". false.
    false):
document.myForm.mySelect.options[0] = new Option("Обычный"  "norm". false.
    false):
document.myForm.mySelect.options[0] - new Option("Средний". "med". false.
    false):
```

Такой синтаксис подходит для всех браузеров, начиная с Netscape 4 и IE 4, но Navigator 4 не умеет изменять ширину списка, и обычно требуется вызов history.go(0) для того, чтобы новые значения вступили в силу.

Обсуждение

Конструктор объекта option имеет следующий синтаксис вызова:

```
var newOpt = new Option("текст", "значение". isDefaultSelectedFlag.
    isSelectedFlag);
```

Параметр текст представляет собой строку, содержащую текстовое значение, которое пользователь может видеть в списке элемента select. Параметр значение содержит строку, которая будет послана на сервер при выборе этого значения. Далее следуют два булевских параметра, позволяющие определить, будет ли данная опция выбрана по умолчанию (эквивалент атрибута selected в HTML-коде), и будет ли элемент выбран в данное время.

Отдельные опции можно менять, присваивая новые значения элементам массива options. Но если нужно изменить сразу большое количество строк, лучше удалить все старые и начать с пустого списка.

Данные для формирования новых объектов option могут браться из таких источников, как объекты JavaScript, внедренные в страницу. Это означает, что все возможные варианты выбора необходимо включить в сценарий на странице. Кроме того, следует продумать структуры данных, которые будут помещаться в элементы select. В качестве примера можно рассмотреть страницу, содержащую два элемента select, с помощью которых пользователи по всему миру смогут выбирать ближайший к ним город. Первый элемент будет служить для выбора одной из областей. После того как область выбрана, второй список должен предлагать на выбор города, принадлежащие этой области. Вот HTML-код для этих двух элементов:

```
Переслать запрос в: <select name="continent" onchange="setCities(this)">
    <option value="" selected>Выберите область:</option>
    <option value="africa">Африка</option>
    <option value="asia">Азия</option>
    <option value="australia">Австралия/Океания</option>
    <option value="europe">Европа</option>
    <option value="noamer">Северная Америка</option>
    <option value="soamer">Южная Америка</option>
```

```
</select> :
<select name="city">
    <option value="" selected>Выберите город:</option>
</select>
```

Все данные, необходимые функции setCities(), которая изменяет содержимое второго элемента select, хранятся в объекте JavaScript. Вот структура данных, оформленных в виде объекта JavaScript:

```
var regiondb - new Object()
regiondb["africa"] - [{value:"102". text:"Каир"}.
                      {value:"88". text:"Лагос"},
                      (value:"80". text:"Найроби"},
                      {value:"55". text:"Претория"}];
regiondb["asia"] = [{value:"30". text:"Анкара"}.
                    {value:"21". text:"Бангкок"}.
                    (value:"49". text:"Пекин"}.
                    {value:"76". text:"Нью-Дели"},
                    {value:"14". text:"Токио"}]:
regiondb["australia"] - [{value:"64". text:"Сува"}.
                         {value:"12". text:"Сидней"}];
regiondb["europe"] = [{value:"11". text:"Афины"},
                      {value:"35". text:"Франкфурт"}.
                      jvalue:"3". text:"Лондон"}.
                      (value:"15". text:"Мадрид"}.
                      {value:"1". text:"Париж"}.
                      jvalue:"10". text:"Рим"},
                      {value:"6". text:"Стокгольм"}.
                      (value:"97". text:"Санкт-Петербург"}]:
regiondb["noamer"] = [{value:"73". text:"Даллас"}.
                      {value:"71". text:"Лос-Анджелес"},
                      {value:"5". text:"Нью-Йорк"}.
                      {value:"37". text:"Торонто"}];
regiondb["soamer"] = [{value:"65". text:"Буэнос-Айрес"}.
                      (value:"31". text:"Каракас"}.
                      {value:"66". text:"Рио-де-Жанейро"}]:
```

Каждый раз, когда пользователь выбирает один из регионов мира из первого элемента select, его обработчик события onchange запускает следующую функцию, которая перезаполняет строки второго списка select

```
function setCities(chooser) {
    var newElem;
    var where = (navigator.appName - "Microsoft Internet Explorer") ? -1 . null:
    var cityChooser = chooser.form.elements["city"];
    while (cityChooser.options.length) {
        ci tyChooser.remove(0);
    }
    var choice - chooser.options[chooser.selectedIndex].value:
    var db - regiondb[choice]:
    newElem - document.createElement("option");
    newElem.text = "Выберите город:".
    newElem.value = "";
    cityChooser.add(newElem. where):
    if (choice != "") {
```

```
        for (var i = 0; i < db.length; i++) {
            newElem = document.createElement("option");
            newElem.text = db[i].text;
            newElem.value = db[i].value;
            cityChooser.add(newElem, where);
        }
    }
}
```

В качестве альтернативного источника данных при применении Internet Explorer 5 для Windows или Netscape 6 и старше можно взять хранящиеся на сервере данные в формате XML. Такие данные можно загрузить в отдельный виртуальный документ XML (см. рецепт 14.4) и использовать их для формирования новых объектов option.

Вместо применявшегося до сих пор синтаксиса DOM Level 0 можно взять для модификации узлов дерева элементов синтаксис W3C DOM. Можно задействовать его и для изменения элементов option, но различия в реализации между IE (вплоть до версии 6) и W3C DOM (таким, как он реализован в Netscape 6) усложняют задачу. W3C DOM предоставляет элементу select два метода: **add()** и **remove()**, которые несколько упрощают изменение данных для этого элемента. Ниже показана версия функции **setCities()**, использующая W3C DOM. Она учитывает различие между браузерами и берет второй параметр метода add():

```
function setCities(chooser) {
var cityChooser = chooser.form.elements["city"];
// удаление предыдущих настроек
cityChooser.options.length = 0;
    // используем значение chosen как индекс в хэш-таблице regiondb
    var choice = chooser.options[chooser.selectedIndex].value;
    var db = regiondb[choice];
    // добавляем первый элемент, используемый по умолчанию
    cityChooser.options[0] = new Option("Выберите город:", "", true, false);
    if (choice != "") {
       // перебор элементов массива, хранящегося в хэш-таблице, и заполнение
       // списка опций
        for (var i = 0; i < db.length; i++) {
            cityChooser.options[i + 1] = new Option(db[i].text,
              db[i].value);
        }
    }
}
```

Второй параметр метода add() определяет позицию, в которую будет помещен новый элемент. По стандарту W3C DOM этот параметр должен быть либо ссылкой на существующий объект option, либо null. В последнем случае элемент будет добавлен в конец списка. В IE второй параметр необязателен, но в том случае, когда он используется, должен содержать целое число, означающее индекс в массиве options, куда будет помещен элемент. Для того чтобы поместить новое значение в конец списка, в качестве второго параметра следует передать -1. Поэтому вначале функции необходимо определять браузер, чтобы узнать, какое значение нужно передавать в качестве второго значения.

Смотрите также

Создание объектов описывается в рецепте 3.8.

8.14. Перенос данных формы ежду страницами

NN4 IE4

Задача

Необходимо перенести все данные из формы на одной странице в другую.

Решение

В секции обсуждения приведена библиотека stringForms.js, с помощью которой можно преобразовать все данные формы в строку, которую можно передать в другую форму, используя URL или cookie. Функция form2ArrayString() должна получать в качестве первого аргумента ссылку на формы, данные которой будут преобразованы в строку. Эта функция возвращает строку, формат которой соответствует сокращенной форме записи конструктора массива объектов. Каждый из этих объектов соответствует одному из элементов формы и имеет следующий вид:

```
{name:'имяЭлемента',id:'идентификаторЭлемента',elemType:'типЭлемента'
   value:значение}
```

Чтобы распределить по форме с той же структурой элементов данные из массива объектов, используется функция string2FormObj(), параметрами которой являются ссылка на объект формы и строка, ранее сформированная form2ArrayObj().

Обсуждение

В листинге 8.3 показан код библиотеки stringForms.js (применявшейся в рецепте 10.6). Эта библиотека предоставляет две функции, которые должны вызываться сценарием: form2ArrayString(), преобразующая данные формы в строку, оформленную как массив объектов JavaScript, и функция string2FormObj(), которая помещает данные в элементы, имеющие те же идентификаторы.

Листинг 8.3. Библиотека stringForms.js

```
// Определение имени, идентификатора, типа и значения элемента формы,
// которые необходимы функции form2ArrayString()
function formObj2String(obj) {
    var output = "{":
    if (obj.name) j
        output += "name:'" + obj.name +. . . .:
    }
    if (obj id) {
```

продолжение

Листинг 8.3 (продолжение)

```
        output += "id:'" + obj.id + "'.";
    }
    output += "type:'" + obj.type + "'
    switch (obj.type) {
        case "radio":
            if (obj.name) {
                obj - document.forms[0].elements[obj.name];
                var radioVal - "value:false.index:-l":
                for (var ı - 0; i < obj.length; i++) {
                    if (obj[i].checked) {
                        radioVal - "value:true.index:" + i:
                        i - obj.length;
                    }
                }
                output +- radioVal;
            } else {
                output += "value:" + obj.checked;

            break;
        case "checkbox":
            output += "value:" + obj.checked:
            break ;
        case "select-one":
            output += "value:" + obj.selectedIndex;
            break;
        case "select-multiple":
            output += "value:" + obj.selectedIndex;
            break;
        case "text":
            output += "value:'" + escape(obj.value) +.....;
            break;
        case "textarea":
            output +- "value:'" + escape(obj.value) +.....;
            break;
        case "password":
            output +- "value:'" + escape(obj.value) + .....;
            break:
        case "hidden":
            output +- "value:'" + escape(obj.value) +.....;
            break;
        default:
            output +- "";
    }
    output +- "}"
    return output:
}

// Преобразование переданной ссылки в строку, имеющую формат
// массива объектов JavaScript
function form2ArrayString(form) {
    var el em. lastName = "":
    var output = "[":
    for (var i - 0: i < form.elements.length; i++) {
        elem - form.elements[i];
```

```
        if(elem.name && (elem.name != lastName)) {
            output += form0bj2String(form.elements[i]) +
            lastName = elem name;
        }
    output = output.substring(0, output.length-1) + "]";
    return output:
}
// Распределение значений по элементам формы, имеющим подходящие
// имена или идентификаторы
// of the orig nal form controls
function string2Form0bj(form. str) {
    var elem, objArray - eval(str);
    for (var i - 0; i < objArray.length; i++) {
        elem - (objArray[i].name) ? form.elements[objArray[i].name] :
            document.getElementById(objArray[i].id);
        switch (objArray[i].type) {
            case "radio":
                if (objArray[i].name && objArray[i] value &&
                    objArray[i].index >= 0) {
                    elem - elem[objArray[i].index];
                }
                elem.checked - objArray[i].value;
                break:
            case "checkbox":
                elem.checked = objArray[i].value;
                break;
            case "select-one":
                elem.selectedIndex - objArray[i].value;
                break:
            case "select-multiple":
                elem.selectedIndex - objArray[i].value;
                break:
            default:
                elem.value - unescape(objArray[i].value);
```

Функция **form2ArrayObj()** управляет процессом преобразования данных формы в строку путем перебора всех элементов формы. Для каждого элемента она вызывает вспомогательную функцию **formObj2String()**, выполняющую всю основную работу этой библиотеки по преобразованию. Задача функции **formObj2String()** — преобразовать описание элемента, ссылка на который передана в функцию, в строку, оформленную как конструктор объекта в сокращенной форме. Каждый из формируемых таким путем объектов имеет четыре свойства: name, id, type и value. При преобразовании в строку учитываются все элементы управления, воспринимающие пользовательский ввод, и чтение данных из которых не приведет к нарушению политики безопасности.

Итак, ваши сценарии должны вызывать функцию form2ArrayString() для того, чтобы получить строковое описание элементов управления формы. Получая ссылку на форму, находящуюся на текущей странице, эта функция возвращает

строку, которую можно передавать либо как аргументы поиска в URL, либо сохранив ее в cookie, который может прочитать страница с того же домена и сервера.

Когда наступит пора применить полученные данные к форме, следует использовать функцию из этой библиотеки string2FormObj(). Ей необходимо передать ссылку на форму в текущей странице и строку, оформленную как конструктор массива. Для преобразования строки в настоящий массив служит функция eval(), несмотря на то что обычно ее следует избегать по соображениям производительности. Затем элементам формы, имеющим те же имена (а если имя отсутствует — то те же идентификаторы), присваиваются значения, сохраненные в строке. Прочие элементы формы свое значение не меняют.

Хотя данная библиотека учитывает то, что несколько переключателей имеют одно и то же имя, ее функции не рассчитаны на сохранение множественного выбора в вариации элемента select, позволяющей выбирать несколько строк. Это связано с тем, что библиотека использует для определения значения элемента select свойство selectedIndex. Можно изменить библиотеку так, чтобы ее функции учитывали возможность одновременного выбора нескольких элементов, сохраняя в виде строки те опции, для которых свойство selected равно true.

Как продемонстрировано в рецептах 10.4 и 10.6, два из трех способов передачи данных между страницами требуют того, чтобы данные были представлены в виде строки. Поэтому здесь очень удобна сокращенная запись конструктора JavaScript, благодаря чему получаются весьма компактные строки и уменьшается шанс исчерпать предел в 512 символов на URL или ограничение в 1 Кбайт текста на один cookie.

Смотрите также

Дополнительная информация о сокращенном синтаксисе конструкторов массивов и объектов дается в рецептах 3.1 и 3.8. Преобразование объектов в строки, использующее похожую методику, продемонстрировано в рецепте 3.13. Передачу данных между страницами с использованием cookie можно увидеть в рецепте 10.4, с применением аргументов URL — в рецепте 10.6.

9 События

9.0. Вступление

Без событий не было бы динамического HTML. Наверное, это слишком сильно сказано, но для рассмотренных объектных моделей документов это действительно так. Чтобы страница обрела динамизм, нужен выключатель, который бы им управлял.

Есть два класса таких выключателей: действия пользователя и действия системы. Действия пользователя — результат деятельности клавиатуры и мыши. Действия системы, которые более точно можно описать как результат функционирования браузера и документа, происходят при возникновении какого-либо изменения в работе браузера, например, когда документ полностью загружается с сервера или браузер не может загрузить данные для изображения.

Одна из задач автора сценариев — определить, какое событие какого элемента будет запускать отдельные части сценария. Например, со всеми кнопками обычно связаны некоторые сценарии (кроме тех случаев, когда основное действие этой кнопки — отсылка формы на сервер или ее очистка). Менее очевидным примером являются сценарии, проверяющие вводимые в текстовое поле символы и пропускающие только те, ввод которых разрешен. Очень часто события, вызываемые мышью, используются для того, чтобы менять изображение на элементе, подсвечивая его. При этом соответствующее событие используется для того, чтобы восстановить изображение, когда указатель мыши уйдет в сторону.

События и сценарии

С точки зрения написания сценариев, нужен способ связать событие и функцию, которая выполняет действия по его обработке. Про события говорят, что они привязаны к элементам. Благодаря этому можно указать, чтобы обработчик определенного события вызывался только для элемента, который это событие вызвал. Например, можно заставить кнопку реагировать на щелчок мышью на ней, в то время как все окружающие элементы щелчки по ним будут игнорировать.

Синтаксис, используемый для связывания событий и их обработчиков, может иметь самый разный вид (см. рецепт 9.1), но обычно автору сценария необходимо просто назначить функцию обработчиком определенного события определенного

элемента. Именно такая функция и называется обработчиком события. Так, если обработчиком события click кнопки заранее назначена функция handleClick(), эта функция запускается при каждом щелчке на кнопке.

Можно устанавливать обработчики для сколь угодно большого числа событий произвольного числа элементов. Так, можно указать картинке менять свое изображение на подсвеченное, когда пользователь проводит курсор над элементом img, а в ответ на щелчок пользователем на кнопке — менять изображение на вогнутое. После того как кнопка мыши отпущена, изображение должно восстановиться. В этом случае четырем разным событиям элемента img следует назначить разные функции (это соответственно события mouseover, mousedown, mouseup и mouseout).

Когда при написании сценария необходимо обратиться к событию, обычно используется имя события с префиксом on. Так, если кнопка получает событие click, для того чтобы установить для него обработчик, необходимо использовать имя onclick.

Типы событий

Знание различных событий может существенно повлиять на ваши возможности создавать интерактивные страницы. Некоторые типы событий доступны, начиная с браузеров версии 4. Такие события играют роль общего знаменателя, так как не зависят от конкретного браузера и конкретной операционной системы. Они перечислены в табл. 9.1, в которой также указана их совместимость с различными браузерами. В последних браузерах обработчик события может быть назначен практически для любого отображаемого элемента, в то время как в более ранних версиях (особенно в IE 3 и Navigator 4) можно использовать только некоторый набор основных элементов.

Таблица 9.1. События в DHTML

Событие	NN	IE/Win	IE/Mac	HTML	Описание
onabort	3	4	3.01	—	Пользователь прервал передачу изображения клиенту
onblur	2	3	3.01	4	Элемент потерял фокус ввода, так как пользователь щелкнул на другом элементе
onchange	2	3	3.01	4	Элемент потерял фокус, при этом его содержимое было изменено с тех пор, как фокус был получен
onclick	2	3	3.01	4	Пользователь нажал и отпустил кнопку мыши (или ее клавиатурный эквивалент) на элементе
ondblclick	4	4	3.01	4	Пользователь сделал двойной щелчок на элементе
onerror	3	4	4	—	Произошла ошибка сценария или ошибка при загрузке внешних данных

Событие	NN	IE/Win	IE/Mac	HTML	Описание
onfocus	2	3	3.01	4	Элемент получил фокус ввода
onkeydown	4	4	4	4	Пользователь начал нажатие клавиши на клавиатуре
onkeypress	4	4	4	4	Пользователь нажал и отпустил клавишу на клавиатуре
onkeyup	4	4	4	4	Пользователь отпустил клавишу
onload	2	3	3.01	4	Завершена загрузка документа или другого элемента
onmousedown	4	4	4	4	Пользователь нажал кнопку мыши
onmousemove	4	4	4	4	Пользователь убрал указатель мыши с элемента
onmouseout	3	3	3.01	4	Пользователь передвинул указатель мыши (вне зависимости от состояния кнопок мыши)
onmouseover	2	3	3.01	4	Пользователь передвинул указатель мыши на элемент
onmouseup	4	4	4	4	Пользователь отпустил кнопку мыши
onmove	4	3	4	—	Пользователь передвинул окно браузера
onreset	3	4	4	4	Пользователь щелкнул на кнопке сброса формы (кнопка Reset)
onresize	4	4	4	—	Пользователь изменил размер окна
onselect	2	3	3	4	Пользователь выделяет текст в текстовом элементе ввода
onsubmit	2	3	3.01	4	Форма отправляется
onunload	2	3	3.01	4	Документ выгружается из содержащего его окна или фрейма

Помимо событий, поддерживаемых разными браузерами, браузеры от Microsoft поддерживают дополнительный набор событий, которые позволяют сценарию реагировать на более специфические действия пользователя и системы. В табл. 9.2 перечислены события, поддерживаемые только IE, которые могут облегчить разработку приложения DHTML. Большинство из них поддерживаются только в версии IE для Windows из-за глубокой интеграции IE в операционную систему. Не указаны в этой таблице только многочисленные события IE, используемые исключительно в технологии связывания данных, которая позволяет привязать элементы формы к источнику в базе данных.

Понимание всех этих событий и их отличительных особенностей — сложная задача. Если же нужно погрузиться в пучины таинственных событий, необходим более подробный справочный материал (например, Dynamic HTML: The Definitive Reference, издан O'Reilly).

Таблица 9.2. События, поддерживаемые только Internet Explorer

Событие	IE/Win	IE/Mac	Описание
onbeforecopy	5	—	Пользователь вызвал команду Копировать, но она еще не выполнена
onbeforecut	5	—	Пользователь вызвал команду Вырезать, но она еще не выполнена
onbeforepaste	5	—	Пользователь вызвал команду Вставить, но она еще не выполнена
onbeforeprint	5	—	Пользователь вызвал команду Печать, но она еще не выполнена
oncontextmenu	5	—	Пользователь вызвал контекстное меню (то есть щелкнул правой кнопкой мыши)
Oncopy	5	—	Пользователь вызвал команду Копировать, но она еще не выполнена
Oncut	5	—	Пользователь вызвал команду Вырезать, но она еще не выполнена
ondrag	5	—	Пользователь перетаскивает элемент
ondragend	5	—	Пользователь завершил перетаскивание элемента
ondragenter	5	—	Пользователь перетащил элемент в область текущего элемента
ondragleave	5	—	Пользователь вытащил элемент из области, занимаемой текущим элементом
ondragover	5	—	Пользователь перетаскивает элемент над текущим
ondrop	5	—	Пользователь бросил перетаскиваемый элемент на текущий элемент
onfocusin	б	—	Событие происходит перед тем, как элементу будет передан фокус (перед onfocus)
onfocusout	б	—	Событие происходит пред передачей фокуса другому элементу (перед onblur)
onhelp	4	—	Пользователь нажал клавишу F1 или выбрал из меню пункт Помощь
onmouseenter	5.5	—	Пользователь переместил указатель мыши на текущий элемент
onmouseleave	5.5	—	Пользователь убрал указатель мыши с элемента
onmousewheel	б	—	Пользователь вращает колесико мыши
onmoveend	5.5	—	Позиционируемый элемент завершил свое движение
onmovestart	5.5	—	Позиционируемый элемент начал свое движение
onpaste	5	—	Пользователь вызвал команду Вставить, но она еще не выполнена
onscroll	4	4	Пользователь передвинул полосу прокрутки у элемента
onselectstart	4	4	Пользователь начал выделять элемент

Модели событий

Ключ к успешной работе с событиями — умение использовать абстрактный объект информацией о событии, который создается каждый раз, когда браузер регистрирует событие. У объекта события имеется множество свойств, в деталях описывающих событие, таких как идентификатор элемента, получившего событие, координаты мыши, состояние клавиатуры, а также тип события. Назначение такого объекта — передать в обработчик необходимую информацию о событии. Например, если обработчик события onmousemove служит для перемещения элемента с помощью мыши, нужно знать координаты мыши при каждом срабатывании события.

К сожалению, ситуацию осложняет то, что модель события, определяющая, каков должен быть объект события и как он должен себя вести, различается в разных браузерах. Что касается Internet Explorer 6 для Windows и IE 5.x для Macintosh, то в них еще не реализована модель W3C DOM Level 2. В то же время Netscape 6 и более поздние версии следуют модели W3C DOM, для удобства поддерживая некоторые из свойств IE. Каждая модель использует свой способ передачи объекта события функции-обработчику (в рецепте 9.1 показано, как решить эту задачу). Кроме того, может потребоваться привести к одному виду свойства объекта события для разных браузеров, чтобы одна функция могла работать в обоих мирах. В табл. 9.3 представлен сравнительный обзор двух объектных моделей.

Таблица 9.3. Эквивалентные свойства объектов событий в IE и NN 6

Свойство в IE	Описание	Метод или свойство W3C
altKey	Состояние клавиши Alt (булевское свойство)	altKey
button	Кнопка мыши, нажатая в момент возникновения события (целое число; используются разные системы нумерации)	button
cancelBubble[1]	Должно ли событие передаваться дальше	stopPropagation()
clientX, clientY	Координаты курсора мыши в момент возникновения события относительно контентной области окна браузера	clientX, clientY
ctrlKey	Состояние клавиши Ctrl (булевское свойство)	ctrlKey
fromElement	Для события mouseover и mouseout — элемент, с которого пришел курсор	relatedTarget
keyCode	Код символа для клавиатурного события	keyCOde
offsetX, offsetY	Координаты курсора мыши относительно элемента, вызвавшего событие	(вычисляются из других свойств)
(вычисляются из других свойств)	Координаты относительно документа	pageX, pageY
returnValue	Значение, возвращаемое системе из обработчика	preventDefault()

продолжение

[1] Поддерживается для удобства в браузерах, основанных на Mozilla

Таблица 9.3 (продолжение)

Свойство в IE	Описание	Метод или свойство W3C
screenX, screenY	Координаты курсора относительно экрана	screenX, screenY
shiftKey	Состояние клавиши Shift (булевское свойство)	shiftKey
srcElement	Элемент, который вызвал событие	target
toElement	Объект или элемент, на который переместился указатель (для событий mouseover и mouseout)	relatedTarget
type	Название события (без префикса on)	type
x, y	Координаты события относительно элемента body, если элемент-источник не позиционируемый, в противном случае — относительно самого элемента	layerX, layerY

Несколько рецептов из этой главы демонстрируют, как можно выровнять большинство наиболее часто используемых свойств в модели IE и W3C DOM.

9.1. Выравнивание модели IE и W3C DOM

NN4 **IE4**

Задача

Необходимо, чтобы сценарий мог получать информацию о событии, возникшем на странице, загруженной как в Internet Explorer, так и в браузер, поддерживающий только модель W3C DOM.

Решение

Так как Internet Explorer (по крайней мере, версия 6 для Windows и 5.x для Macintosh) не поддерживает модель W3C DOM, а Netscape 6 и старше поддерживает только ее, для упрощения дальнейшей обработки к свойствам следует использовать свои способы для каждой модели.

Обработчики событий по стандарту W3C DOM получают ссылку на объект события через параметр (смотрите обсуждение), в то время как в IE объект event является свойством объекта window. Чтобы учесть оба этих варианта и сформировать для дальнейших вычислений единую ссылку, необходимо в каждой функции-обработчике использовать такую структуру:

```
function functionName(evt) {
    evt = (evt) ? evt : ((window.event) ? event : null);
    if (evt) {
        // Дальнейшая обработка события
    }
}
```

В этой структуре вместо определения версии браузера применяется проверка объектов, что позволяет не только получить ссылку, специфическую для каждой объектной модели, но и избежать ситуации, когда браузер не поддерживает ни одну из моделей (то есть значительно более старые браузеры, реагирующие на простые события).

Обсуждение

Чтобы в браузере, использующем модель событий W3C DOM, передать объект события в обработчик, требуется либо выполнить небольшое предварительное планирование, либо вообще ничего не предпринимать, в зависимости от способа, который был выбран для связывания обработчика с элементом. Формально спецификация W3C DOM поддерживает только один способ назначения функции-обработчика: через метод **addEventListener(),** имеющийся у любого элемента. Вот пример его использования:

```
document.getElementById("myButton").addEventListener("click",
    processClick, false);
```

Этот метод был разработан для W3C DOM и впервые реализован в Netscape 6. Здесь функция, вызываемая при срабатывании события, автоматически получает объект события в качестве единственного аргумента. Так что достаточно определить значение этого аргумента и назначить его переменной в начале функции (в показанном выше примере значение назначается переменой **evt).**

К счастью для разработчиков, вынужденных поддерживать несколько браузеров, два способа назначения обработчиков событий, предшествующих стандарту W3C DOM, без всяких проблем работают в Netscape 6 и будут работать еще долгое время. Первый способ — назначение обработчика путем присваиванием значения специальному свойству элемента, второй — указание обработчика через атрибуты тега.

Назначение обработчика через свойство объекта выполняется одним оператором JavaScript. В левой части этого оператора стоит ссылка на элемент и имя свойства, представляющее собой имя события с префиксом on. В правой части оператора присваивания стоит ссылка на функцию, которая будет вызвана в момент получения события элементом. Здесь ссылка на функцию представляет собой просто имя функции, без круглых скобок:

```
document.getElementById("myButton").onclick = processClick;
```

Если обработчик назначен таким образом, в браузерах, основанных на Mozilla, функция обработчика автоматически получает единственный аргумент, ссылку на объект события.

Другой механизм назначения обработчиков, поддерживаемый многими браузерами, — назначение обработчиков через атрибуты, внедренные в тег элемента. В Интернете можно увидеть множество примеров кода, использующего такой механизм, потому что это был изначальный способ связывания, появившийся вместе с самими сценариями. К тому же, данный метод предоставляет удобный способ передать один или несколько произвольных аргументов функции-обработчику. Так, нередко можно увидеть, что в формах элементы **input** передают в обработ-

чик ссылку на себя (this) или ссылку на форму (this.form), чтобы обработчик мог выполнять действия со значением элемента или с другими элементами управления на форме. Чтобы в функцию обработчика, привязанную к событию таким образом, передать объект события W3C DOM, список аргументов должен включать ключевое слово event:

```
<input type="button" name="myButton" id="myButton" value="Process"
onclick="processClick(event)">
```

Можно передавать в функцию и несколько аргументов, при этом порядок аргументов при вызове функции должен совпадать с порядком, в котором они объявлены. Например, если обработчик события описан так:

```
onclick="processClick(this.form, event)"
```

функция обработчика должна быть объявлена как функция с двумя аргументами, чтобы перехватить второй переданный ей параметр:

```
function processClick(form, evt) {     ]
```

Если ссылка на объект события явно передается в обработчик, как в показанном выше примере, IE при выполнении этого кода помимо нее передаст и свою собственную версию объекта event, так что такой синтаксис будет выполнять двойную работу. Но в развитии языков сценариев и разметки документа наблюдается тенденция отказаться от подобного связывания через теги с целью разделения содержимого, оформления и обработки.

С другой стороны, назначение обработчиков иначе, чем через теги, имеет свои недостатки. Самый главный из них — невозможность назначить обработчик до того, как объекты будут полностью загружены. JavaScript пытается разрешать ссылки на элементы в момент присваивания, и если при этом объект не будет известен интерпретатору, присваивание приводит к ошибке выполнения сценария. Это верно даже для элемента body: в некоторых браузерах попытка назначить обработчик события элементу document. body из сценария, расположенного в головной секции страницы, приводит к ошибке.

Чтобы решить эту головоломку, можно поместить все присваивания обработчиков событий (или почти все) в отдельную функцию, которая будет вызываться однократно после того, как страница целиком загрузится. Дополнительные подробности можно увидеть в рецепте 9.2.

Если вы используете показанную в этом решении технологию приведения объектов событий к одному виду, то вам, возможно, понадобится делать дополнительное выравнивание в зависимости от платформы. Причина в том, что нужное свойство в разных браузерах может иметь разное имя (примеры можно увидеть в рецептах 9.3 и 9.6). Но между моделями объекта события в разных браузерах достаточно много сходства, поэтому вполне допустимо применить единую функцию-обработчик для всех браузеров.

Смотрите также

В табл. 9.3, приведенной во вступлении к этой главе, имеется сравнительный перечень свойств объектов событий для разных браузеров, все прочие рецепты в этой книге используют методику приведения к единому виду. В рецепте 9.2

показано, как запускать функцию (для назначения обработчиков событий) сразу после загрузки страницы. Выравнивание различий в названиях отдельных свойств объектов события демонстрируется в рецептах 9.3 и 9.6.

9.2. Инициализация после загрузки страницы

NN2 **IE3**

Задача

Сразу после загрузки новой страницы в браузер необходимо запустить функцию, для выполнения которой требуется, чтобы все содержимое, включая изображения и программные модули, а также элементы управления ActiveX, было загружено и готово к работе.

Решение

Событие onload объекта window возникает строго после того, как все его содержимое загрузится и инициализируется. Поэтому это событие можно использовать для запуска функции, необходимой для завершения инициализации объектов. Показанный ниже синтаксис работает во всех браузерах, поддерживающих выполнение сценариев:

```
<body onload="myImtFunction()">      </body>
```

Кроме того, во всех браузерах, начиная с Navigator 3 и Internet Explorer 4, можно использовать следующую форму записи; такой сценарий может быть помещен в головной секции документа:

```
winodw.onload = myInitFunction;
```

При использовании последнего варианта записи нужно следить, чтобы функция myInitFunction() была объявлена выше в коде сценария.

Обсуждение

В многих рецептах из этой книги используются одна или несколько функций инициализации, вызываемых обработчиком события onload. Здесь предпочтение отдается первому варианту записи, когда вызов обработчика события помещается в атрибут элемента body. При этом вызов всегда легко найти, обычно он располагается поблизости от основного кода сценария. Если же необходимо применить несколько функций инициализации, как, например, в библиотеке DHTML API из рецепта 13.3 и в некоторых других модулях, можно совмещать вызов инициализаторов в правильной последовательности (если имеется взаимное влияние между модулями):

```
<body onload="initDHTMLAPI(); setHeights()">
```

При назначении обработчика через свойство невозможно назначить несколько функций на одно событие, так как при попытке назначить несколько обработчиков действует только самое последнее назначение. Поэтому если нужно, чтобы по событию window.onload вызывалось несколько инициализирующих функций, обработчик события должен в таком случае передавать управление специальной управляющей функции, которая и вызывает отдельные инициализаторы:

```
function startup() {
    initDHTMLAPI();
    setHeights();
}
```

Установка всех обработчиков событий только после полной загрузки страницы имеет одно дополнительное преимущество: события не будут обрабатываться до тех пор, пока страница полностью не загрузится. Это особенно полезно, если страница использует сценарии для взаимодействия с элементами управления ActiveX или с подключаемыми программными модулями. Дело в том, что если пользователь щелкнет на кнопке, вызывающей сценарий до того, как все вспомогательные структуры загрузятся, произойдет ошибка сценария.

Использование обработчика события onload для запуска сценария, назначающего обработчики остальных событий, имеет несколько преимуществ. Прежде всего, очень старые браузеры автоматически не будут реагировать на события, так как они не понимают назначение событий через свойства. Тем самым отсеиваются браузеры, которые не способны воспринять обработчики, рассчитанные на более современные браузеры. Во-вторых, можно ограничить выполнение сценариев, используя методику проверки объектов. Например, если нужно, чтобы функция обработчик выполнялась только на браузерах, поддерживающих синтаксис обращения к элементам W3C DOM, необходимо окружить операторы, устанавливающие обработчик, условными операторами, проверяющими выполнение нужных требований:

```
function initEvents() {
    // Эти обработчики нормально работают во всех браузерах.
    // кроме NN2 and IE3
    document.forms[0].onsubmit - validateMainForm;
    document.forms[0].elements["email"].onchange - validateEmail;
    if (document.getElementById) {
        // А эти работают только при поддержке W3C DOM
        document.getElementById("logo").onclick - goHome;
        document.body.onclick = blockEvent;
    }
}
window.onload = initEvents:
```

такой подход во многом превосходит оборачивание блоков кода сценария тегами <script> с указанием версии интерпретатора в атрибуте language (то есть language= "JavaScript1.2"). Использовать версию JavaScript для выяснения поддерживаемой объектной модели довольно опрометчиво. Вместо этого следует применить проверку объектов.

Смотрите также

В обсуждении рецепта 9.1 перечислены способы назначения обработчиков событиям.

9.3. Определение координат мыши

NN6 IE5

Задача

Необходимо определить координаты щелчка мыши (или другого события) относительно всего документа или относительно элемента, получившего событие.

Решение

В рецепте представлены два решения для разных ситуаций, потому что в каждом варианте по-разному объединяются координаты мыши и координаты страницы, обычно используемые для позиционирования элементов. Пусть пользователь щелкает мышью где-либо на странице, указывая, куда следует поместить позиционируемый элемент. Представим себе, что пользователь щелкнул на карте, чтобы расположить на ней изображение стрелки. Возникает вопрос: нужно ли производить позиционирование относительно всей страницы или относительно прямоугольника, который занимает позиционируемый элемент? Используя две функции, которые описаны в обсуждении, getPageEventCoords() и getPositionedEventCoords(), можно получить координаты, которые соответствуют координатам события. Обе эти функции возвращают объект, свойства left и top которого соответствуют координатам события.

Основа пользовательского интерфейса в этом примере — две версии функции moveToClick(), применяющей функцию shiftTo() из DHTML API (см. рецепт 13.3). Когда пользователь щелкает мышью там, где воспринимаются события, удерживая при этом клавишу Shift, позиционируемый элемент перемещав гся так, чтобы его левый верхний угол оказался в указанной точке.

В первой рассматриваемой ситуации используются координаты относительно пространства, занимаемого самой страницей, что позволяет позиционировать в указанную пользователем точку элемент первого уровня (то есть элемент, не являющийся вложенным). Обработчик события можно подключить к элементу document:

```
document.onmousedown - moveToClisk;
```

В этой версии функция **moveToClick()** использует функцию **getPageEventCoords()**. Возвращенное ей значение используется как аргумент для shiftTo().

```
function moveToClick(evt) {
    evt = (evt) ? evt : event;
    if (evt.shiftKey) {
        var coords = getPagetEventCoords(evt);
        shiftTo("mapArrow", coords .left, coords.top):

}
```

Во втором варианте позиционирования по щелчку мыши задача состоит в том, чтобы расположить вложенный подвижный элемент внутри родительского. Другими словами, необходимо определить координаты щелчка мышью относительно внешнего подвижного элемента, так как внешний элемент определяет собственную координатную плоскость для вложенного в него. В этой ситуации лучше всего подключить обработчик события к внешнему подвижному элементу, хотя это и не обязательное требование. Это выполняется в процедуре инициализации, запускаемой событием **onload:**

```
document.getElementById("myMap").onmousedown = moveToClick;
```

Здесь moveToClick() вызывает функцию getPositionedEventCoords(), для определения координат:

```
function moveToClick(evt) {
    evt = (evt) ? evt : event;
    if (evt.shiftKey) {
        var coords = getPositionedEventCoords(evt);
        shiftTo("mapArrow", coords.left, coords.top);
    }
}
```

Обсуждение

В функции getPageEventCoords(), показанной далее, имеется две основные ветви, необходимые для определения положения события мыши в координатной плоскости документа. Первая из них применяет более простые в применении свойства, имеющиеся в Netscape: pageX и pageY. Но в IE для того, чтобы точно переместить элемент в указанное место, приходится выполнять гораздо больше вычислений. Используя свойства **clientX** и **clientY,** нужно учитывать прокрутку документа и небольшой сдвиг, который IE автоматически добавляет к документу (обычно это два пиксела по обеим осям). Если же IE работает в режиме совместимости с CSS, в уравнение следует внести небольшой сдвиг элементов HTML.

```
function getPageEventCoords(evt) {
    var coords = {left:0, top:0};
    if (evt.pageX) {
        coords.left = evt.pageX;
        coords.top = evt.pageY;
    } else if (evt.clientX) {
        coords.left =
            evt.clientX + document.body.scrollLeft -
          document.body.clientLeft;
        coords.top =
            evt.clientY + document.body.scrollTop - document.body.clientTop;
        // если нужно, учитываем положение внешнего элемента
        if (document.body.parentElement &&
          document.body.parentElement.clientLeft) {
            var bodParent = document.body.parentElement;
            coords.left += bodParent.scrollLeft - bodParent.clientLeft;
            coords.top += bodParent.scrollTop - bodParent.clientTop;
        }
    }
    return coords-
}
```

Определение координат события относительно подвижного элемента — задача функции **getPositionedEventCoords()**, код которой показан в следующем листинге. Здесь ветка IE, которая поддерживает свойства **offsetX** и offsetY, самая простая. Ветка Netscape использует свойства **layerX** и **layerY,** значения которых требуется только немного подстроить, чтобы учесть границы элемента. А чтобы предотвратить дальнейшую передачу этого события (что может привести к конфликтам с другими событиями mousedown), его свойству **cancelBubble** присваивается значение **true:**

```
function getPositionedEventCoords(evt) {
    var elan = (evt.target) ? evt.target : evt srcElement;
    var coords = {left:0, top:0};
    if (evt.layerX) {
        var borders = {left:parseInt(getElementStyle("progressBar",
                        "borderLeftWidth", "border-left-width")),
                       top:parseInt(getElementStyle("progressBar"
                        "borderTopWidth", "border-top-width"))};
        coords.left = evt.layerX - borders.left;
        coords.top = evt.layerY - borders.top;
    } else if(evt.offsetX) {
        coords.left = evt.offsetX;
        coords.top = evt.offsetY;
    }
    evt.cancelBubble = true;
    return coords;
}
```

Кроме того, следует учесть еще одну проблему, связанную с совместимостью. Если у документа с помощью таблицы стилей установлена рамка, то Netscape и IE (в любом из режимов) расходятся во мнении о том, начинается ли система координат элемента с его рамки или с содержимого. Netscape включает рамку в систему координат, а IE — нет. Таким образом, чтобы уравнять обе ситуации, нужно в случае Netscape учитывать ширину рамки. Это делается с помощью функции **getElementStyle()** из рецепта **11.12:**

```
function getElementStyle(elemID, IEStyleAttr, CSSStyleAttr) {
    var elem = document.getElementById(elemID);
    if (elem.currentStyle) {
        return elem.currentStyle[IEStyleAttr];
    } else if(window.getComputedStyle) {
        var compStyle = window.getComputedStyle(elem, "");
        return compStyle.getPropertyValue(CSSStyleAttr);
    }
    return "";
}
```

Может показаться странным, что определение координат становится в некоторых случаях столь трудоемким. Отчасти это объясняется тем, что разработчики стандартов объектов событий и систем координат документа даже не представляли, как DHTML-разработчики будут использовать эти возможности. Объектная модель W3C DOM лишь немного помогает разработчикам, определяя две пары свойств, относящихся к системе координат: **clientX/clientY** и **screenX/screenY.** Но даже тогда формальное описание свойств **clientX и clientY** оставляет большую

свободу интерпретации. Понимается ли под «клиентской областью» документа весь документ целиком или только его видимая часть? В Netscape это понятие интерпретируется как весь документ, в то время как в IE свойствах **clientX** и **clientY** измеряются от границы видимой части документа (что, как известно, не соответствует стандарту W3C D0M), поэтому здесь требуется поправка.

Стандарт W3C DOM умалчивает о координатах событий в позиционируемых элементах. Конечно, с помощью некоторых вычислений эти координаты можно вычислить, исходя из свойств **clientX** и **clientY,** а также настроек стиля элемента. Свойства **offsetX/offsetY,** имеющиеся в **IE,** и layerX/layerY в Netscape несколько упрощают дело, но и они не совсем идеальны.

Даже при использовании поправок, показанных в примерах этого рецепта, может встретиться такое сочетание стилей рамок, границ и отступов, которое разрушит все эти аккуратные подсчеты. Если подобное CSS-оформление относится к элементу body позиционируемого объекта, вероятно, придется поэкспериментировать с поправками, которые бы работали в ситуациях, встречающихся на страницы. В частности, можно определить значения свойств offsetLeft, offsetTop, clientLeft и clientTop не только того элемента, с которым ведется работа, но и тех, которые могут влиять на его положение (обычно они доступны через свойство offsetParent). Также не стоит упускать из виду настройки рамок, границ и отступов CSS, которые могут повлиять на систему координат элемента. Нужно выяснить, какие значения определяют количество потерянных пикселов. Это утомительный процесс, так что приготовьтесь потратить на него некоторое время.

Смотрите также

Блокирование передачи событий демонстрируется в рецепте 9.4. Вспомогательная функция, определяющая настройки импортируемой таблицы стилей, приведена в рецепте 11.12. Определение пиксельных координат элемента в документе смотрите в рецепте 13.8.

9.4. Блокирование событий

NN6 **IE5**

Задача

Необходимо блокировать работу события, выполняемую по умолчанию.

Решение

Хотя попытки использовать сценарии для того, чтобы помешать пользователю, скажем, щелкнуть правой кнопкой мыши на изображении, чтобы сохранить его на жесткий диск, бесплодны, в этом рецепте описывается, как помешать отдельным посетителям, которых может остановить предупреждающее диалоговое окно.

Основное событие, которое нужно блокировать в этом случае, oncontyextmenu, которое реализовано в IE 5 для Windows и в Netscape 6. Для решения этой задачи следует назначить обработчик этого события у всего документа. Показанная ниже функция блокирует это событие у всех элементов img:

```
function blockEvents(evt) {
    evt - (evt) ? evt : event:
    var blockit - false:
    var elem - (evt.target) ? evt.target : ((evt.srcElement) ?
       evt.srcElement . null):
    if (elem && elem.tagName && elem.tagName.toLowerCase() == "img") {
        if (evt.cancelBubble) {
            evt.cancel Bubble = true;
        }
        alert("Нет доступа."):
        return false;
    }
document.oncontextmenu = blockEvents:
```

Обсуждение

Запрет обычного поведения может придать вашим страницам возможности, которые они сами по себе не могли бы иметь. Например, если в текстовое поле скучно вводить только числа, можно проверять сведения события и блокировать нечисловые данные. Аналогичным образом можно блокировать событие **submit** при проверке данных формы на стороне клиента (см. главу 8).

События можно было блокировать и до того, как в браузерах появилась современная сложная модель события. Основа этой методики — убедиться, что в последней строке обработчика события стоит оператор, эквивалентный

```
return false:
```

Например, в рецепте 5.10 показывается, как элемент может ссылаться на один документ, если браузер не поддерживает сценарии, и открывать другой, если сценарии поддерживаются:

```
<a href="std/newProducts.html" title="Новые продукты"
  onclick="return linkTo('ieEin/newProducts.html', 'w3/newProducts.html')">
Новые продукты</a>
```

Здесь функция **linkTo()** возвращает значение **false,** благодаря чему это же значение возвращает и обработчик события **onclick.** Поэтому элемент a никогда не будет использовать ссылку href, если разрешено выполнение сценариев, потому что с точки зрения элемента щелчок на нем никогда не произойдет. Обработчик события можно также оформить в виде двух отдельных операторов:

```
onlcick="linkTo('ieEin/newProducts.html', 'w3/newProducts.html'):
    return false"
```

Первый вариант записи идеален для ситуации, когда функция-обработчик на основании переданных ей данных решает, должно ли событие достигнуть элемента. В рецепте 8.3 показано, как использовать эту методику для проверки данных формы, а в рецепте 8.11 демонстрируется, как ограничить набор символов, которые можно вводить в текстовое поле.

По мере усложнения модели события старые способы блокирования продолжают работать (и работают до сих пор), но сейчас используются способы связывания событий и обработчиков, которые не позволяют напрямую включать оператор return в обработчик. В том случае, если обработчик назначается через соответствующее свойство, для управления блокированием события следует применить последний оператор функции. Так, если последний оператор возвращает true, событие обрабатывается как обычно, а если false, нормальное поведение отменяется.

Эта техника была синтаксически улучшена в объектной модели IE, где объект event имеет свойство returnValue. Этому свойству можно присваивать значения true или false для того, чтобы указать элементу, нужно ли выполнять обычные действия. Дополнительным плюсом такой методики является то, что можно возвращать из функции что-либо отличное от булевского значения, блокирование события управляется независимо.

Похожие возможности имеются и в модели W3C DOM, где у объекта event, передаваемого в обработчик, имеется специальный метод. Чтобы отменить обычные действия, выполняемые событием, следует вызвать метод preventDefault() без параметров. В Netscape 6 и более поздних версиях вызов этого метода эквивалентен присвоению значения false своиству returnValue в IE4. Конечно, если требуется совместимость с разными браузерами, необходимо использовать старый стиль блокирования с использованием return false.

Рассмотрим процесс распространения событий при их обработке. Модель событий в IE и W3C DOM позволяет событиям передаваться вверх по дереву элементов документа. Например, событие click одной из кнопок формы вызывается в самой кнопке, в элементе form, в элементе body и в корневом элементе document. Это можно применять для того, чтобы использовать один обработчик события для нескольких однотипных элементов на странице. Для этого следует разрешить событиям от элементов передаваться в содержащий их элемент, которому и назначить обработчик нужного события.

Такое распространение события от элементов нижнего уровня к верхним иногда называется «всплыванием» события (то есть событие как бы всплывает вверх по дереву элементов). Если обработчик события присоединен к элементу, стоящему выше в иерархии документа, может потребоваться блокировать передачу событий от тех элементов, чьи сообщения не должны обрабатываться. В противном случае их события будут также вызывать обработчик в элементе более высокого уровня и запутывать вычисления.

Для управления распространением событий в IE у объекта event имеется свойство cancelBubble. Это же свойство реализовано для большей совместимости и в Netscape 6 (хотя оно не является частью стандарта W3C DOM Level 2). Если присвоить этому свойству значение true, оно будет распространяться. В W3C DOM такая возможность реализуется с помощью метода объекта event stopPropagation(). Пока что он реализован только в Netscape 6 и более поздних версиях.

Вот вариант функции blockEvents(), пересмотренной с учетом специфики IE:

```
function blockEvents() {
    var blockit - false:
    var elem = event.srcElement;
    if (elem && elem.tagName && elem.tagName toLowerCase() == "img") {
```

```
        event.cancelBubble = true;
        event.returnValue = false;
        alert("Нет доступа.");
    }
```

При использовании чистого синтаксиса W3C DOM функция принимает вид:

```
function blockEvents(evt) {
    var blockit = false;
    var elem = (evt.target) .
    if (elem && elem.tagName && elem.tagName.toLowerCase() == "img") {
        evt.preventDefault();
        evt.stopPropagation();
        alertC'HeT доступа.");
    }
}
```

Несмотря на все попытки сетевых разработчиков, не существует способа помешать пользователю просматривать исходный код страницы (в том числе и подключаемые библиотеки JavaScript, а также таблицы стилей). Также невозможно помешать пользователю сохранить копию изображения, показанного на странице. В то же время это не только основная причина блокирования событий, но и важное требование многих авторов информационного наполнения. Любая техника, основанная на использовании сценария, моментально терпит неудачу, как только пользователь запретит выполнение сценариев в своем браузере. Даже если пытаться усложнить задачу, открывая документ в окне браузера без строки меню, открываемый URL можно узнать и из основного документа, а значит, можно открыть его вручную в обычном окне. Несмотря на слабый уровень такой «защиты от правой кнопки», множество разработчиков, наблюдавших подобную защиту на других страницах, думают, что она предлагает настоящую защиту. Часто они интересуются в сетевых формах, как реализовать то, что они видели. Это дает поистине анекдотическое доказательство того, что даже опытных разработчиков можно отпугнуть, используя несложный трюк.

Смотрите также

Другие рецепты, где используется управление распространением событий: блокирование гиперссылок используется в рецепте 5.10, блокирование отсылки данных формы — в рецепте 8.3, текстовые поля рассматриваются в рецепте 8.11.

9.5. Блокирование двойного щелчка

NN3 **IE3**

Задача

Необходимо предотвратить повторное срабатывание щелчка на кнопке или гиперссылке.

Решение

Для этого в обработчике события onclick следует выполнить необходимые действия, а затем переназначить обработчик этого события. Например, показанная ниже ссылка отсылает данные формы, используя для этого функцию submitForm():

```
<a href="#" onclick="return submitForm( )">Готово</a>
```

В самом обработчике события нужно после выполнения основного действия назначить новый обработчик, который ничего не будет делать:

```
function submitForm( ) {
    document.forms["myForm"].submit( );
    submitForm = blockIt;
    return false;
}
function blockIt( ) {
    return false;
}
```

Обсуждение

Обратите внимание на то, что в этом рецепте не упоминается обработчик события ondblclick. Причина в том, что данное событие не имеет отношения к такому блокированию. Если не назначать этому событию никакого обработчика, при его появлении в системе все равно ничего не произойдет. Здесь же необходимо предотвратить повторный вызов обработчика. Это особенно важно при отправке форм в приложениях электронной коммерции. Если пользователь повторно нажмет кнопку отправки, пока страница все еще видима, сервер может обработать обе отсылки, в результате чего заказ будет сохранен в базе данных дважды.

Показанная в решении методика обманчиво проста. Второй оператор функции submitForm() подменяет ее функцией blockIt(). Поэтому при повторных вызовах submitForm() вместо нее выполняется код функции blockIt(). Помните, что такое переназначение функции будет действовать до тех пор, пока страница не будет перезагружена.

Если же нужно создать сценарий, который выполнится при двойном щелчке на элементе, можно использовать событие ondoubleclick. Но попытка одновременно обрабатывать как событие onclick, так и событие ondoubleclick одного и того же элемента может дать неудовлетворительные результаты. В реальных приложениях для этого служит комбинация обработчиков ondoubleclick и onmousedown.

Смотрите также

Блокирование обычного поведения события смотрите в рецепте 9.4.

9.6. Определение элемента, получившего событие

NN6 **IE4**

Задача

Необходимо добыть ссылку на элемент, получивший последнее событие.

Решение

Объектная модель события как в IE, так и в Netscape предлагает свойства, с помощью которых можно определить, какой элемент изначально получил событие. Несмотря на различия в синтаксисе, можно привести свойства **srcElement** в **IE и target** в стандарте W3C DOM к одному виду, чтобы получить действительную ссылку во всех браузерах. Использование этой методики требует приведения объектов события к одному виду, что показано в рецепте 9.1.

Типичная функция обработчика события, корректно работающая в старых браузерах, обычно начинается так:

```
function myFunction(evt) {
    evt = (evt) ? evt : ((window.event) ? event   null):
    if (evt) {
       var elem = (evt.target) ? evt.target   ((evt.srcElement) ?
           evt.srcElement : null):
       if (elem) {
           // Здесь выполняется обработка события
       }
    }
```

Обсуждение

Модель события W3C DOM, в том виде, в каком она реализована в Netscape, приводит к некоторым дополнительным сложностям в том случае, когда элемент является контейнером для одного или нескольких текстовых узлов. По стандарту W3C DOM такие узлы сами могут быть источниками событий. Таким образом, если назначить обработчик события **onclick** элементу, который заключает в себе текст с гиперссылкой, переданный в обработчик объект event будет ссылаться на гиперссылку, вложенную в контейнер. Тем не менее обработчик наверняка более заинтересован в ссылке на элемент-контейнер и в его свойствах. Поэтому если элементы, события которых обрабатываются, могут служить контейнерами, следует изменить шаблон обработчика из этого рецепта следующим образом:

```
function myFunction(evt) {
    evt = (evt) ? evt : ((window.event) ? event : null):
    if (evt) {
```

```
        var elem = (evt.target) ? evt.target  ((evt srcElement) ? evt.srcElement • null)
        if (elem nodeType ==3) {
            elem = elem.parentNode;
        }
        if (elem ) {
            // Здесь выполняется обработка события
        }
    }
}
```

Ссылка на элемент, получивший событие, не всегда нужна, однако показанная методика полезна в тех случаях, когда обрабатываются события элементов, требующие некоторой связи с контекстом. Пусть текстовому полю назначается обработчик события onchange с помощью следующего синтаксиса:

```
document.getElementById("emailAddress").onchange = validateEmail;
```

В этом случае функция для выполнения работы нуждается в дополнительной информации об элементе. В показанном ниже примере используются свойства поля ввода value и form:

```
function validateEmail(evt) {
    evt = (evt) ? evt   ((window.event) ? event : null);
    if (evt) {
        var elem = (evt.tagret) ? evt.target:
            ((evt.srcElement) ? evt.srcElement : null);
        if (elem.nodeType == 3) {
            elem = elem.parentNode:
        }
        if (elem) {
            var val = elem.value;
            var form = elem.form;
            // Здесь производятся прочие действия no обработке события
        ]
    }
}
```

Если сценарий пишется в расчете только на объектную модель W3C DOM, можно воспользоваться преимуществами, которые дает комбинация распространения событий и свойства currentTarget. Это свойство возвращает ссылку на узел, с которым связан обработчик события. Например, если элемент a (гиперссылка) заключает в себе текстовый узел, можно назначить этому элементу обработчик события, из которого можно будет с помощью свойства currentTarget получить ссылку на сам элемент, несмотря на то, что источником события является текстовый узел:

```
function handleLinkClick(evt) {
    var elem = evt.currentTarget.
    // Обработка событий элемента <a>
}
```

Сейчас наблюдается тенденция переносить обработчики событий из тегов, поэтому значение методик определения ссылки на получивший событие элемент все возрастает.

Смотрите также

В рецепте 9.1 показано, как привести к одному виду объекты событий в IE и W3C DOM.

9.7. Определение нажатой кнопки мыши

NN6 **IE4**

Задача

Необходимо, чтобы обработчик события мог определить, какая кнопка мыши (или какая комбинация кнопок) привела к возникновению события.

Решение

Обе объектные модели IE и W3C DOM единогласно считают, что информацию о нажатой кнопке или сочетании клавиш, вызвавших событие mousedown, должно хранить свойство button (события mouseup и click не всегда предоставляют информацию о кнопках). К сожалению, на этом сходство заканчивается.

В разных моделях целочисленные константы, связанные со свойством button, различаются. Правда, обе модели согласны с тем, что значение 2 означает правую (вторичную) кнопку. Вот пример кода функции, которая различает нажатие на правую кнопку и на все остальные:

```
function myFunction(evt) {
    evt = (evt) ? evt . ((window.event) ? event . null);
    if (evt) {
        if (evt.button) {
            if (evt.button = 2) {
                // обработка нажатия на правую кнопку
                return false:
            } else {
                // все остальные кнопки обрабатываются здесь
            }
        }
    }
```

Обсуждение

Причина, из-за которой в показанной функции производится проверка наличия свойства button, — необходимость корректной работы в Netscape 4, который для хранения информации о кнопке использует совершенно другое свойство и числовые константы. Это свойство называется which, правой кнопке соответствует значение 3. Чтобы учесть это различие в том случае, если необходима поддержка Netscape 4, можно переработать структуру функции таким образом:

```
function myFunction(evt) {
    evt = (evt) ? evt : ((window.event) ? event : null);
```

```
    if (evt) {
        var rightButton = false;
        if ((evt.button && evt.button == 2) ||
           (evt.which && evt.which - 3)) {
            rightButton = true;
        }
        if (rightButton) {
            // обработка нажатия на правую кнопку
            return false;
        } else {
            // все остальные кнопки обрабатываются здесь
        }
    }
}
```

Опять же, нельзя забывать, что единственное событие, для которого можно с уверенностью определить, какая кнопка нажата, — это mousedown. Для всех остальных событий вместо информации о кнопках доставляется значение 0 (в IE) или null (в NN).

Все возможные значения, которые может принимать свойство button, перечислены в табл. 9.4. Наиболее всесторонняя система имеется в IE.

Таблица 9.4. Возможные значения свойства button

Кнопка	IE 4+	NN 6+	W3C DOM
Нет кнопки	0	null	null
Левая (основная)	1	0	0
Средняя	4	1	1
Правая	2	2	2
Левая + Правая	3	–	–
Левая + Средняя	5	–	–
Правая + Средняя	6	–	–
Левая + Средняя + Правая	7	–	–

Если в целевую группу пользователей входят пользователи Macintosh, необходимо постараться ограничить использование правой кнопки мыши, применяя ее только для некритичных действий. На этих компьютерах по традиции используются мыши с одной кнопкой, хотя в будущем ситуация может измениться. Хотя некоторые браузеры и могут эмулировать нажатие правой кнопки, когда щелчок производится с удержанием одной из клавиатурных клавиш-модификаторов, нельзя гарантировать, что пользователи знают, как это сделать. Таким образом, представляя пользователям Windows и Unix возможность выполнять какое-нибудь действие с помощью правой кнопки, следует позаботиться об альтернативе для пользователей Mac. Можно также обеспечить возможность выполнять определенное действие щелчком мыши с удержанием определенной клавиши на клавиатуре для пользователей всех операционных систем.

Смотрите также

В рецепте 9.1 показано, как привести к одному виду объекты событий в IE и W3C DOM.

9.8. Считывание нажатого символа

NN4 **IE4**

Задача

Необходимо проверять нажатия всех буквенно-цифровых клавиш до того, как символ достигнет поля формы.

Решение

Показанную ниже функцию следует назначить обработчиком события onkeypress для одного из полей формы, в котором нужно ограничить набор допустимых символов цифрами от 0 до 9, знаком минус и запятой:

```
function numberOnly(evt) {
    evt = (evt) ? evt · ((window.event) ? event   null):
    if (evt) {
       var elem = (evt.target) ? evt.target   ((evt.srcElement) ?
           evt srcElement : null):
       if (elem) {
           var charCode = (evt.charCode) ? evt.charCode :
               ((evt.which) ? evt.which · evt.keyCode):
           if ((charCode < 32 ) ||
               (charCode > 44 && charCode < 47) jj
               (charCode > 47 && charCode < 58)) {
               return true:
           } else {
               return false:
           }

    }
}
```

Обсуждение

Из трех клавиатурных событий: onkeydown, onkeyup и onkeypress, для проверки нажатой клавиши используется onkeypress. Использование двух остальных событий смотрите в рецепте 9.9.

Информация о нажатой клавише хранится в объекте события в различных свойствах, имена которых зависят от используемой объектной модели. В IE 4 и позже информация обо всех клавиатурных событиях хранится только в свойстве keyCode. В ответ на событие onkeyPress это свойство содержит код нажатого символа в кодировке Unicode. В Netscape 6 свойство keyCode зарезервировано Для других целей, а для хранения кода нажатого символа используется свойство

charCode. Устаревшая модель событий Netscape 4 использовала для хранения кода свойство which (то же, что и для хранения номера кнопки мыши в относящихся к мыши событиях). В предыдущем примере все три системы приводятся к одному виду, и определяется единственное значение, которое и можно применять для дальнейших сравнений.

Для символов английского языка кодировка Unicode совпадает с кодировкой ASCII. Символы и их ASCII-коды показаны в приложении А. Обратите внимание на то, что строчные и прописные символы имеют разные коды; свои коды есть и у стандартных знаков пунктуации. В дополнение к этому, некоторые управляющие клавиши имеют коды в нижней части таблицы (Tab, Space, Backspace и Enter).

Основная сложность функции numberOnly() заключена в комбинации условных выражений в операторе if. Суть этих сравнений в том, чтобы допускать символ до поля ввода лишь в том случае, если его код попадает в одну из следующих категорий:

- код символа меньше, чем наименьший код печатного символа (это символ пробела, имеющий код, равный 42);
- код равен 45 или *AA* (коды знака минус и запятой);
- символ — одна из цифр от 0 до 9.

Все остальные символы приводят к тому, что функция возвращает значение false, из-за чего событие не достигает текстового поля.

В IE (как для Win, так и для Mac) имеется возможность изменять значение свойства объекта события, хранящего код нажатого символа. Некоторые могут посчитать, что такая возможность таит в себе потенциальную угрозу безопасности: ничего не подозревающий пользователь набирает текст, а сценарий символ за символом вводит совершенно другой текст. Тем не менее эта возможность весьма удобна при разработке формы для базы данных, которой требуется, чтобы все передаваемые строки были в верхнем регистре. Можно автоматически преобразовывать символы нижнего регистра в символы верхнего, оставляя остальные символы неизменными. Функция (которую можно использовать только в IE) следующая:

```
function upperOnly() {
    var charCode = event.keyCode;
    if (charCode > 96 && charCode < 123) {
        event.keyCode = charCode - 32;
```

Идея подхода, работающего в различных браузерах, такова: использовать обработчик события onchange, в котором прогонять строку из поля ввода через метод объекта String toUpperCase(), и помещать результат обработки обратно в поле.

Смотрите также

События, вызываемые клавишами, отличными от символьных, рассматриваются в рецепте 9.9. В рецепте 9.10 показано, как определить состояние клавиатурных модификаторов при нажатии на клавишу. Предварительное фильтрование информации, вводимой в текстовое поле, более подробно рассматривается в рецепте 8.11.

9.9. Клавиши, отличные от символьных

NN6 **IE4**

Задача

Запустить сценарий при нажатии пользователем клавиши, отличной от буквенно-цифровой.

Решение

Для определения кода, соответствующего нажатию такой клавиши, используется одно из событий onkeydown или onkeyup. Код хранится в свойстве keyCode объекта **event** (в IE 4 или NN 6 и старше). Ниже показана функция, которая при нажатии курсорных клавиш сдвигает абсолютно позиционируемый элемент на пять пикселов в соответствующем направлении. Этот пример использует функцию **getElementStyle(),** описанную в рецепте **11.12.**

```
function handleArrowKeys(evt) {
    evt = (evt) ? evt : ((window.event) ? event  null);
    if (evt) {
        var top = getElementStyleC'moveableElem". "top". "top");
        var left = getElementStyleC'moveableElem". "left". "left");
        var elem = document.getElementById("moveableElem");
        switch (evt.keyCode) {
            case 37:
                el em.style.left - (parseInt(left) 5) + "px":
                break:
            case 38:
                el em.style.top = (parseInt(top) - 5) + "px":
                break;
            case 39:
                el em.style.left = (parseInt(left) + 5) + "px"
                break.
            case 40:
                el em.style.top = (parseInt(top) + 5) + "px".
                break;
        }
    }
}

document.onkeyup = handleArrowKeys:
```

В этом примере задействовано событие **onkeyup,** так как в Netscape 7 обработка события **onkeydown** производится с ошибкой (а в Netscape 6.x не работают ни **onkeyup,** ни **onkeydown).** Это плохо, так как в IE, если держать клавишу нажатой до тех пор, пока не начнется ее автоповтор, автоматически будет повторяться и событие onkeydown, продвигая подвижный элемент в нужном направлении (такое поведение является желательным для пользовательского интерфейса).

Обсуждение

Коды клавиш отличаются от кодов символов. С каждой физической клавишей на клавиатуре ассоциирован некоторый код. Например, клавише 2 на основной клавиатуре и клавише 2 на цифровой соответствуют символы с одинаковым кодом, но коды символов, передаваемые в обработчики onkeyup и onkeydown при нажатии на эти клавиши, различаются. Кроме того, коды буквенно-цифровых клавиш не зависят от того, удерживалась ли при их нажатии клавиша Shift. Коды клавиш для типичной английской клавиатуры перечислены в приложении В.

Разработка приложения, использующего специальные клавиши, — весьма коварная задача, так как каждый браузер и каждая операционная система по-своему определяют нормальное поведение таких вещей, как функциональные клавиши и клавиши навигации. Чтобы сценарий не могло полностью заблокировать приложение, браузеры не позволяют блокировать стандартные для браузера или операционной системы сочетания клавиш. Так, в показанном выше примере курсорные клавиши использовались для управления подвижным элементом. Если при этом остальная страница не будет помещаться на экране, то помимо перемещения подвижного элемента можно будет наблюдать обычную прокрутку страницы. При этом, сколько ни возвращай из обработчика false и сколько ни блокируй передачу события, отменить обычное поведение будет невозможно. (Хотя в данном случае можно попробовать переместить фокус в поле ввода, в котором нажатия курсорных клавиш не приводят к прокрутке страницы.)

Если для выполнения какого-либо действия планируется использовать функциональную клавишу, можно делать это только в тех браузерах и операционных системах, поведение которых предсказуемо. Даже в среде Windows следует быть уверенным, что функциональная клавиша действительно не выполняет никаких действий, даже таких обычно незаметных, как перемещение по строкам меню. Некоторые же функциональные клавиши выполняют действия, вообще не имеющиеся ни в одном из меню. Все, что остается, — проверять, проверять и еще раз проверять.

Предупреждения, касающиеся функциональных клавиш, относятся и к попыткам использовать сочетания клавиш. Нельзя изменить поведение стандартного сочетания клавиш, поэтому поиск сочетаний, не занятых ни в одном из браузеров, может стать сложной задачей. В следующем примере показан обработчик события onkeyup, вызывающий функцию runSpecial() при нажатии сочетания клавиш Ctrl+Alt+P в IE 4, Netscape 7 и более поздних версиях.

```
function handleAccelerator(evt) {
    evt = (evt) ? evt · ((window.event) ? event : null):
    if (evt) {
        if (evt.keyCode == 80 && evt.ctrlKey && evt.altKey) {
            runSpecial();
        }
    }
}

document.onkeyup = handleAccelerator;
```

При связывании сценария с сочетанием клавиш лучше всего использовать обработчик события у узла document, так как этот узел является верхним в иерархии. Все клавиатурные события, возникшие во вложенных элементах (таких

как текстовые поля и поля ввода), будут переданы в этот узел, если только их передача не будет отменена где-то по пути. Если событие произошло в одном из текстовых полей, объект события будет содержать ссылку на это поле Благодаря этому можно создать контекстно-зависимую справку, вызываемую комбинацией Ctrl+Alt+F2. Эта справка будет предоставлять дополнительную информацию о поле, которое редактирует пользователь:

```
function showHelp(elem) {
    var elemID - elem.id:
    switch (elemID) {
        case "name".
            alert("Введите свое полное имя.");
            break;
        case "email":
            alert("Вам будет передан ваш код доступа  \n" +
                  "Убедитесь, что адрес введен точно и он действует.");
            break,

    }
}

function handleAccelerator(evt) {
    evt = (evt) ? evt . ((window.event) ? event : null):
    var elem = (evt.target) ? evt.target : ((evt.srcElement) ?
       evt srcElement : null);
    if (evt) {
        // для комбинации Ctrl+Alt+F2
        if (evt.keyCode == 113 && evt ctrlKey && evt.altKey) {
            showHelp(elem);
        }
    )
}
document.onkeyup = handleAccelerator;
```

Здесь нельзя использовать клавишу F1, поскольку в IE для Windows эта клавиша зарезервирована для запуска справочной системы.

Смотрите также

Как привести разнородные объектные модели события к одному виду, показано в рецепте 9.1. Определение символа, соответствующего нажатой клавише, показано в рецепте 9.8. Как определить состояние клавиш-модификаторов, описывается в рецепте 9.10.

9.10. Ctrl, Alt и Shift

NN6 **IE4**

Задача

Необходимо узнать, были ли при возникновении события нажаты клавиши Ctrl, Alt или Shift.

Решение

В IE и стандарте W3C DOM для определения состояния клавиш-модификаторов служит один и тот же набор свойств объекта события. Вот имена этих свойств:

- altKey;
- ctrlKey;
- shiftKey.

Имеется еще одно свойство, metaKey, которое соответствует клавише Command на клавиатурах Macintosh (но это свойство не поддерживается IE/Mac). Каждое из перечисленных свойств содержит при вызове события одно из значений true или false. Если свойство равно true, соответствующая клавиша была нажата в момент возникновения события. Ниже показан пример обработчика события, выполняющего один набор действий при щелчке без модификаторов и другой, если удерживать клавишу Shift:

```
function handleClick(evt) {
    evt = (evt) ? evt : ((window.event) ? event : null);
    if (evt) {
        if (evt.shiftKey) {
            // обработка Shift+Click
        } else {
            // обработка немодифицированного щелчка
        }
```

Обсуждение

Реализовать обработку только тех событий, которые произошли при нужном состоянии клавиатурных модификаторов, так же просто, как и добавить еще одну дополнительную проверку в обработчик события. Например, если нужно, чтобы обработчик события выполнял блок кода только в том случае, если нажаты клавиши Shift и Ctrl, следует использовать такую запись:

```
if (evt.shiftKey && evt.ctrlKey) {      ]
```

Разрабатывая многочисленные варианты действий, можно легко увлечься. Но с точки зрения удобства пользовательского интерфейса лучше ограничить число предлагаемых пользователю вариантов, так как он легко может запутаться в том, какое действие за что отвечает. Пользователи имеют различный опыт использования клавиатурных модификаторов в других программах, поэтому можно рассмотреть такой вариант, когда предлагается всего одна альтернатива, которая выполняется при нажатии любой из клавиш-модификаторов. Чтобы добиться таких действий, необходимо использовать булевский оператор ИЛИ (||):

```
if (evt.altKey || evt.ctrlKey || evt.metaKey || evt.shiftKey) {...}
```

Что касается использования свойства metaKey в Netscape на Macintosh, следует иметь в виду следующие сведения (особенно если у вас нет опыта работы с Mac). В программах под Macintosh имеется поддержка комбинаций клавиш того же рода, что и в других операционных системах, но если в Windows повсеместно

используются сочетания с клавишей Ctrl, на Macintosh обычно применяется клавиша Command. Другими словами, то, что пользователю Windows или Unix удобно делать с помощью Ctrl, пользователю Mac удобнее делать, используя Command. Также в приложениях и управлении пользовательским интерфейсом часто используется клавиша Alt, которая на Macintosh называется Option (например, при перетаскивании значков на рабочем столе, чтобы поместить в папку копию файла)

Смотрите также

Обработка событий мыши рассматривается в рецепте 9.7. Считывание значения нажатого символа демонстрируется в рецепте 9.8. Рецепт 9.9 посвящен работе с функциональными клавишами и клавишами навигации.

9.11. Определение элемента под курсором

NN6 **IE5**

Задача

Необходимо выяснить, на какой элемент переместился курсор, а также какой элемент он покинул.

Решение

Модель события в IE поддерживает два разных свойства: fromElement и toElement, которые содержат ссылки на соответствующие элементы. Модель же W3C DOM предлагает только одно свойство, relatedTarget, выполняющее обе задачи. То, на какой элемент передается в этом случае ссылка, зависит только от события. Событие mousemove приводит к тому, что в это свойство помещается ссылка на элемент, с которого пришел курсор, а событие mouseout — элемент, на который переместился курсор. Например, следующий элемент вызывает различные обработчики, в зависимости от того, передвинулся курсор с элемента или на него:

```
<img src="myImg.jpg"   . onmouseover="incoming(event)"
onmouseout=outcoming(event)" />
```

Показанная ниже функция incoming() приводит свойства события к единому виду, получая ссылку на элемент, с которого передвинулся курсор:

```
function incoming(evt) {
    evt = (evt) ? evt • ((window.event) ? event  null):
    if (evt) {
        var from = (evt relatedTarget) ? evt.relatedTarget :
            ((evt.fromElement) ? evt.fromElement • null):
        if (from) {
            // работа с соседним элементом, с которого передвинулся курсор
        }
    )
```

Параллельная ей функция outgoing() формирует ссылку на элемент, на который перешел курсор:

```
function outgoing(evt) {
    evt = (evt) ? evt : ((window.event) ? event  null);
    if (evt) {
        var to = (evt.relatedTarget) ? evt.relatedTarget
            ((evt.toElement) ? evt.toElement : null);
        if (to) {
            // работа с соседним элементом, на который передвинулся курсор
        }
    }
}
```

Обсуждение

Рассмотрим более конкретный пример взаимодействия между элементами с использованием событий. На рис. 9.1 показана таблица, в которой есть одна центральная клетка и четыре «горячих» клетки с каждой ее стороны. Если переместить курсор на центральную клетку, страница покажет, с какой клетки он передвинулся. Если убрать курсор с центральной клетки, надпись покажет, куда он передвинулся.

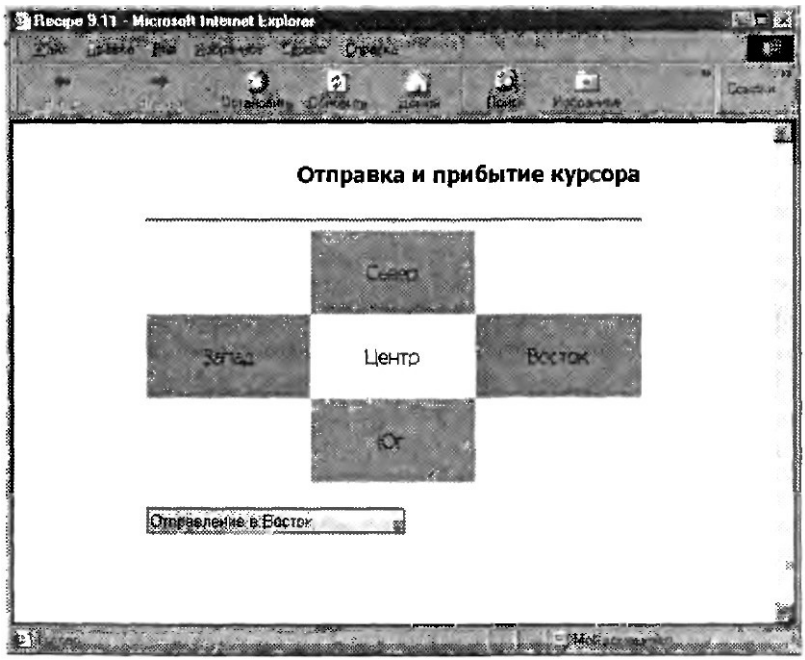


Рис. 9.1. Внешний вид таблицы

Вот HTML-код для таблицы, текстового поля и обработчиков события для центральной клетки:

```
<table cellspacing="0" cellpadding="25">
<tr><td></td><td class="direction">Север </td><td></td></tr>
```

```
<tr><td class="direction">Запад </td>
<td id="main" onmouseover="showArrival(event)"
              onmouseout="showDeparture(event)">Центр </td>
<td class="direction">Восток </td></tr>
<tr><td></td><td class="direction">Юг </td><td></td></tr>
</table>

<form name="output">
<input id="direction" type="text" size="30" />
</form>
```

Таблица стилей задает цвета клеток, чтобы можно было отличать одну от другой:

```
<style type="text/CSS">
.direction {background-color:#00ffff;
            width:100px;
            height:50px;
            text-align:center
           }
#main {background-color:#fff6666; text-align:center}
</style>
```

Две следующие функции считывают соответствующие свойства объекта event и отображают результаты своей работы в текстовом поле:

```
function showArrival(evt) {
    var direction = "";
    evt = (evt) ? evt : ((window.event) ? event : null);
    if (evt) {
        var elem = (evt.target) ? evt.target : ((evt.srcElement) ?
            evt.srcElement : null);
        if (elem) {
            // обрабатываем только элементы
            if(elem.nodeType == 1) {
                // используем свойство W3C DOM
                if (evt.relatedTarget) {
                    if (evt.relatedTarget != elem.firstChild) {
                        direction = (evt.relatedTarget.firstChild) ?
                            evt.relatedTarget.firstChild.nodeValue :
                            "неизвестные края";
                    }
                // используем свойства IE DOM
                } else if (evt.fromElement) {
                    direction = (event.fromElement.innerText) ?
                        event.fromElement.innerText : "неизвестные края";
                }
                // отображение результатов
                document.getElementById("direction").value = "Прибытие из: "
                  + direction;
            }
        }
    }

function showDeparture(evt) {
    var direction = "";
    evt = (evt) ? evt : ((window.event) ? event : null);
    if (evt) f
```

```
    var elem = (evt.target) ? evt.target  ((evt srcElement) ?
        evt.srcElement : null):
    if (elem) {
        // отрабатываем только элементы
        if (elem.nodeType == 1) {
            // используем свойство W3C DOM
            if (evt.relatedTarget) {
                if (evt.relatedTarget != elem.firstChild) {
                    direction = (evt.relatedTarget.firstChild) ?
                        evt.relatedTarget.firstChild.nodeValue :
                        "неизвестные края ";
                }
            // используем свойство IE DOM
            } else if (evt.toElement) {
                direction = (event.toElement.innerText) ?
                    event.toElement.innerText  "неизвестные края "
            }
            // отображаем результаты
            document.getElementById("direction").value = "Отправился в:
              + direction;
        }
    }
}
)
```

Так как в объектной модели W3C рассматриваются события для текстовых узлов, показанные выше функции должны ограничиться обработкой элементов td, содержащих текстовые подписи. Если не сделать этого, событие mouseover текстового узла передастся вышележащему элементу td (если, конечно, у текстового узла нет обработчика события, блокирующего передачу) и свойство relatedTarget будет ссылаться на сам центральный элемент td. Другими словами, в событии, которое получает текстовый узел, окружающий его элемент td будет считаться элементом, с которого пришел курсор. Это один из тех случаев, когда не получается автоматически получить ссылку на родительский элемент для текстового узла (обратный случай смотрите в рецепте 9.3), так как свойство relatedTarget события имеет разные значения для узлов двух разных типов.

На то, насколько успешным окажется применение тех или иных функций объекта события, может оказывать и сложность тех элементов, с которыми ведется работа. Особенно это относится к модели событий W3C DOM, но не ограничивается только ею. Если элемент-контейнер содержит в себе много вложенных элементов без интервалов между ними, достаточных для своевременного срабатывания событий mouseover и mouseout, можно потерять нужные события. Эта проблема возникает, когда пользователь быстро двигает мышь и курсор проскакивает открытую область контейнера, не производя в нем никаких событий.

Перечисленные замечания указывают на то, что применять данную технологию следует в тщательно контролируемом окружении. В идеальной ситуации должны быть большие элементы, граничащие друг с другом и не содержащие вложенных элементов. Например, заполненная изображениями, образующими прямоугольную мозаику таблица будет работать надежно. Если, конечно, вам действительно необходимо получать при обработке событий мыши информацию о смежных элементах.

Смотрите также

В рецепте 9.1 показано, как привести к общему виду различные объектные модели событий.

9.12. Привязка звуков к событиям

NN(нет) **IE5(Win)**

Задача

Необходимо связать звук с определенным событием некоторого элемента.

Решение

Чтобы упростить задачу, это решение работает только в Internet Explorer для Windows, используя его возможность управлять Windows Media Player (Проигрыватель Windows Media). В этом примере сложный HTML-код, в котором следует убедиться, что элементу object указана правильная информация, необходимая для загрузки проигрывателя.

Здесь с элементами раскрывающегося меню (описанного в рецепте 10.8) будет ассоциирована пара коротких звуков. Высокий тон будет означать, что выделен заголовок меню, а более низкий — соответствует одному из пунктов. Так как браузеры не предлагают никаких встроенных звуков, для этого рецепта следует записать или получить иным образом пару коротких звуковых файлов, называемых hi.wav и lo.wav

Начнем с HTML-кода, в который нужно добавить два тега **<object>**, по одному на каждый звук:

```
<object id="hiPing" width="1" height="1"
    classid="CLSID:22d6f312-b0f6-11d0-94ab-0080c74c7e95"
    codebase="#Version=6,0,0,0">
    <param name="FileName" value="hi.wav">
    <param name="AutoStart" value="false">
</object>

<object id="loPing" width="1" height="1"
    classid="CLSID:22d6f312-b0f6-11d0-94ab-0080c74c7e95"
    codebase="#Version=6,0,0,0">
    <param name="FileName" value="lo.wav">
    <param name="AutoStart" value="false">
</object>
```

Затем к сценариям на странице следует добавить следующую функцию:

```
function playSound(id) {
    if (document.all && document.all[id].FileName) {
        document.all[id].Play();
```

Теперь можно проигрывать звук там, где это нужно, вызывая функцию и передавая ей идентификатор звукового объекта, который нужно проигрывать:

```
playSound("hiPing");
```

В случае раскрывающегося меню из рецепта 10.8 необходимо добавить команду playSound("hiPing") в функцию swap() сразу перед вызовом showMenu(). Затем в функцию toggleHighlight() добавляется вызов playSound("lowPing"), непосредственно перед оператором keepMenu().

Обсуждение

В решении показана настройка тега <object>, рассчитанная на использование довольно старой версии проигрывателя Windows Media (6.4), чтобы убедиться в его нормальной работе в IE 5 и старше. Метод Play(), вызываемый для проигрывания звука, совместим и с новыми версиями проигрывателя, но значение атрибута classid приводит к тому, что загружается старая версия. То, что каждому звуку соответствует отдельный тег, помогает предварительно загрузить все звуки вместе со страницей, тем самым уменьшая задержку между первым самым действием и звуком.

Синхронизировать звук с быстрым событием — действительно непростая задача. Прежде всего, звук должен быть очень малой длительности, без плавного нарастания или начальной задержки. Чем короче звук, тем больше шансов, что проигрыватель будет готов к повторному проигрыванию, если пользователь быстро перемещает курсор по пунктам меню. Но даже в этом случае не удается достигнуть прицельно точной синхронизации. Некоторую дополнительную задержку вносит в уравнение и управление проигрывателем. Поэтому лучше вызывать команду playSound() до того, как действие будет в действительности сделано.

Среди разработчиков нет единого мнения по поводу внедрения на странице звуков любого рода. Неуместная музыка, играющая при просмотре сайта, может только оттолкнуть посетителей и помешать им посещать сайт в рабочее время. К звукам нужно относиться так же, как и к возможностям, предоставляемым DHTML, улучшая представление документа для тех пользователей, имеющих необходимое оборудование. Это зачастую означает, что нужно предоставить пользователю возможность отключить звуки (или, еще лучше, начать с отключенными звуками и предоставить кнопку для их включения).

Можно управлять звуками, используя другие программные модули и элементы управления ActiveX, но набор браузеров, поддерживающих такие действия, достаточно ограничен. В IE для Mac (вплоть до версии 5.x) не существует способа организации взаимодействия между сценариями и программными модулями. В браузерах Netscape такая возможность существует (это Navigator 4 для Windows и Mac, Navigator 6 или старше для Windows), но для взаимодействия задействован совершенно иной механизм.

Смотрите также

В рецепте 10.8 показано раскрывающееся меню, которое использовалось в этом примере.

10 Навигация по сайту

10.0. Вступление

Ни одна веб-страница не является (и не должна являться) самодостаточной. Точно так же, как есть способ достигнуть страницы, должно быть несколько путей, ведущих с нее на другие страницы, принадлежащие тому же сайту или внешние. Традиционный, без применения сценариев способ навигации по страницам - использовать гиперссылку, элемент HTML с тегом <a>. Но более сложный дизайн пользовательского интерфейса зачастую требует применения динамического HTML, который может предоставить новые возможности навигации и упростить сам процесс перемещения.

Объект location

В каждом окне или фрейме браузера, поддерживающего сценарии, имеется объект location, свойства которого содержат информацию об адресе загруженной в окно страницы. Это абстрактный объект, то есть объект, не имеющий видимого представления на странице, за исключением, может быть, строки адреса браузера. Но объект location не влияет на текст, видимый в строке адреса до тех пор, пока браузеру не удастся полностью загрузить указанный этому объекту адрес.

Свойства объекта location доступны как для записи, так и для чтения. Отдельные свойства открывают доступ к компонентам URL загруженной страницы (а также к URL целиком). Если бы не существовало никаких ограничений в доступе к этой информации, сценарии могли бы шпионить за действиями пользователя без его ведома. Представим, например, неразборчивый в средствах сайт, который выглядит как поисковая страница Google. На самом деле этот сайт действительно показывает страницу Google в одном из фреймов, а другой фрейм невидим для пользователя. Тогда сценарий в самом наборе фреймов или же в невидимом фрейме мог бы каждые несколько секунд проверять значение объекта location у видимого фрейма, накапливая сведения о действиях пользователя, которые потом мог бы отослать своим владельцам без его ведома и разрешения.

Несмотря на то что иногда удобно знать URL другой страницы или фрейма, опасность вторжения в личную жизнь пользователей заставила производителей браузеров ограничить возможности чтения данных объекта location. Браузеры соблюдают разные типы политик безопасности, которые помогают защищать

личную информацию пользователя. Политика безопасности, применяемая к объекту location, называется политикой общего источника. Если сценарий на странице, которая доставлена с определенного сервера и домена, пытается обратиться к объекту location другой страницы или фрейма, документ на той странице должен принадлежать тому же серверу и домену. Поэтому если пользователь в одном из фреймов переместится на адрес, принадлежащий другому серверу, сценарий при обращении к ней получит отказ и информация об адресе будет для него недоступна.

Частично вследствие многочисленных брешей в системе безопасности Internet Explorer для Windows, Microsoft иногда еще больше ограничивает доступ к объекту location на других страницах. Это приводит к отказу в доступе, даже если страница доставлена с того же сервера. Поэтому надежнее всего обращаться к данным объекта location из сценария, принадлежащего тому же окну, что и сам объект. Как будет видно из некоторых последующих рецептов, есть веские причины так поступать.

Все перечисленные замечания относительно безопасности тем не менее касаются только чтения данных объекта location. Можно совершенно свободно присваивать новые значения его свойствам из других окон.

Передача данных между страницами

В мире веб-приложений очень часто используется модель, по существу, являющаяся системой навигации, основанной на формах, в которой практически каждая страница представляет собой форму, данные которой передаются при переходе к следующей странице. Когда данные отправленной формы достигают сервера, серверное приложение разделяет их на отдельные пары имя/значение. Некоторые из этих пар могут сохраняться в базе данных. Прочие могут быть переработаны в значения, помещенные в невидимые элементы форм на следующей странице. После того как вторая страница будет доставлена, сервер не будет знать, находится ли пользователь на сайте или же давно ушел куда-то еще. Другими словами, сервер только отвечает на запросы браузера, возвращая в ответ на них страницы.

Сервер может быть запрограммирован так, чтобы удерживать некоторую временную информацию о пользователе, узнавая его по идентификатору сессии. Этот идентификатор передается браузеру вместе с каждой страницей, поэтому, когда браузер посылает запрос, сервер может связать запросы одного и того же пользователя друг с другом. Некоторые серверные программы, «на лету» формирующие страницу для каждого пользователя (например, amazon.com), заполняют атрибут href всех ссылок внутри сайта, вписывая туда значения идентификатора сессии для передачи его от страницы к странице. Это может показаться несколько неуклюжим, но благодаря такой технологии передается значительно меньше данных, чем при поддержании постоянного соединения между браузером и сервером (или между сервером и тысячами браузеров по всему миру).

Между тем, не у каждого есть навыки программирования или доступ к серверу, необходимые для использования серверного способа передачи данных от страницы к странице. Доступ к жесткому диску пользователя невозможен по тем же причинам. Но, к счастью, используя JavaScript и фрагменты разных объектных

моделей, можно найти несколько способов передачи данных от страницы к странице, минуя сервер. В рецептах с 10.4 по 10.6 показано, как можно передавать данные через cookie, фреймы и URL. В качестве примера рассмотрим ситуацию, когда пользователь сделал закладку на страницу сайта, расположенную в одном из фреймов, в то время как прочие фреймы содержат важные инструменты навигации. Если пользователь загрузит такую ссылку в браузер, простой сценарий на странице удостоверяется в том, что загружается не только набор фреймов, но и указанная страница в нем, вместо загружаемой по умолчанию.

Всплывающие и раскрывающиеся меню

Раскрывающиеся меню навигации или всплывающие из неподвижных элементов пользовательского интерфейса (например, из строки или панели меню) невероятно экономят пространство. Вместо того чтобы перечислять дюжины вариантов выбора, пользователю показывают только категории верхнего уровня, а проведя мышью над названием категории, пользователь может увидеть список входящих в нее ссылок. Такая концепция интерфейса понятна всем пользователям Windows, Mac и X Window System.

Каждый гуру DHTML делал когда-нибудь систему меню, использующую свойства видимости и позиционирование элементов, работающую в браузерах версии 4 и выше. Вряд ли мир нуждается в еще одной такой системе. Но книга рецептов была бы без нее неполной. Одно из непреодолимых препятствий на пути разработчика — невозможность охватить все ситуации. Каждый разработчик по-своему представляет себе систему меню, поэтому разработка с применением стиля из рецепта может потребовать намного больше усилий, чем новая таблица стилей. Таким образом, цель заключается не в создании полностью универсальной системы меню — основное внимание уделяется созданию переносимого, как можно более простого кода, достаточно гибкого для внесения любых дополнительных изменений (код использует библиотеку DHTML, которая более подробно описана в главе 13).

Прежде чем внедрять всплывающие меню в сайт, особенно в общедоступный, следует убедиться, что эти меню — не более чем элемент дополнительного удобства и не критичны для навигации. Необходимо обеспечить доступ к сайту пользователям, чьи браузеры не поддерживают JavaScript, даже если для этого потребуется дополнительная загрузка страницы. Опираясь на использование традиционных ссылок для навигации без применения сценариев, можно обеспечить корректную работу поисковых машин, которые в этом случае могут добраться до страниц, расположенных в глубине сайта.

Передача данных в страницу

В некоторых рецептах этой и других глав используется набор невидимых для пользователя данных, доступных для сценария на странице. В зависимости от особенностей приложения такие данные могут быть как статическими, так и динамическими. Во втором случае данные извлекаются из базы и преобразовываются в форму, пригодную для применения в сценарии.

До недавних пор существовало не так уж много способов передать данные в страницу. Это массивы или объекты JavaScript, жестко внедренные в HTML-страницы (см. рецепт 14.5) или «на лету» встраиваемые в код страницы, генерируемой сервером. В некоторых новых браузерах, особенно в Internet Explorer 6 для Windows и в браузерах, основанных на Mozilla (например, Netscape 6 и старше), имеется возможность загружать данные XML в невидимые виртуальные документы. Сценарии могут использовать такие данные, привлекая для доступа к ним основанные на DOM-стандартах методики (см. рецепт 14.4). В свете того, что все большее и большее количество информации из баз данных сохраняется и транспортируется в формате XML, последняя методика, несомненно, перспективна. Пройдет какое-то время, прежде чем большая часть типичных браузеров научится загружать данные XML, поэтому не стоит торопиться внедрять эту технологию на общедоступный сайт. Однако в локальных сетях, где используемые браузеры поддерживают XML, этот подход, несомненно, привлекателен.

10.1. Загрузка страницы или якоря

NN2 **IE3**

Задача

Необходимо, чтобы переход к новой странице или якорю был произведен с помощью сценария (вместо обычной ссылки).

Решение

Для того чтобы загрузить в текущее окно или фрейм новую страницу, следует присвоить свойству **location.href** строковое значение, содержащее нужный URL:

```
location.href = "http://www.megacorp.com/products/framistan309.html";
```

Чтобы переместиться на якорь в текущей странице, нужно присвоить свойству **location.hash** строку, содержащую имя якоря (значение, присвоенное тегу в атрибуте name):

```
location.href = "section03";
```

Обсуждение

Значение, присваиваемое свойству location.href, может быть полным или относительным адресом страницы в виде строки. На интерпретацию относительного URL оказывает влияние значение тега <base>, который может находиться на странице (иногда сервер настроен так, чтобы внедрять этот тег во все страницы, тогда он виден в браузере при просмотре HTML). Так как оператор присваивания, меняющий значение location.href, выгружает текущую страницу, нельзя рассчитывать на то, что следующие непосредственно за ним операторы успеют выпол-

ниться до исчезновения всех значений и переменных страницы (хотя некоторые браузеры работают с некоторым опережением).

Поскольку навигация с использованием JavaScript обещала стать весьма популярной, в язык была встроена сокращенная запись доступа к свойствам объекта location. Благодаря этому строковые значения, присваиваемые объекту, помещаются в его свойство href. Таким образом, два показанных ниже выражения выполняют одно и то же действие:

```
location = "someplaceElse.html";
location.href = "someplaceElse.html";
```

Несмотря на кажущееся удобство, лучше использовать полный вариант записи во избежание путаницы в будущем.

Различные устаревшие браузеры поддерживали в той или иной степени и некоторые другие способы навигации, описанные ниже. Данная информация поможет понять значение кода, который можно встретить в некоторых страницах. В дополнение к объекту location для навигации использовался, особенно раньше, объект document. Этот объект более конкретен, чем абстрактный объект location, берущий свое начало из объекта document.

Одно из свойств объекта location вынесено в document. Это свойство document.location, значение которого представляет собой полный или относительный URL документа. Но по причине возможных семантических конфликтов с объектом location, это свойство не рекомендуется к использованию, начиная с третьей версии Navigator. Заместило его доступное для чтения и для записи свойство document.URL, значение которого также представляет собой адрес страницы.

Тем не менее структура W3C DOM сделала свойство document.URL непригодным для навигации (фактически, в спецификации DOM Level 2 нет механизмов навигации), и хотя свойство URL все еще включается в набор свойств объекта HTMLDocument (корневой узел документа HTML), оно описано как свойство только для чтения. Таким образом, можно ожидать, что в будущем это свойство станет недоступным для записи (как в браузерах на основе Mozilla) и тем самым перестанет быть пригодным для навигации.

Наконец, последнее замечание касается метода window.navigate(), который был представлен Microsoft в самых ранних версиях Internet Explorer, и поддерживается до сих пор во всех версиях IE. Единственный параметр этого метода представляет собой строку, содержащую URL. Не стоит использовать этот метод, если нет уверенности, что в будущем ваш сценарий будет использоваться только в IE.

При перемещении к заданному якорю в текущем документе с помощью location.hash включать символ # в строку не нужно, этот символ играет роль разделителя между адресом страницы и именем якоря. Иначе ведет себя свойство location.search, в значение которого следует включать разделительный символ ?, с которого начинается область аргументов URL. Перемещение между якорями в пределах одной страницы должно быть практически мгновенным. Если вы видите, что браузер обращается к серверу каждый раз, когда свойству location.hash присваивается новое значение (особенно это касается IE для Windows), скорее всего, сервер настроен так, что выдает страницы, срок действия которых истекает сразу после

доставки, другими словами, страницы не кэшируются. Если разрешить кэшировать страницы, навигация между якорями должна стать мгновенной.

Смотрите также

Как переходить на разные страницы в зависимости от браузера, показано в рецепте 5.10. Навигация в различных фреймах из набора демонстрируется в рецептах 7.2 и 7.3. Как переместиться из набора фреймов на страницу, не содержащую фреймов, показано в рецепте 7.4.

10.2. Удерживание страницы от попадания в историю браузера

NN3 IE4

Задача

Необходимо удалить текущую страницу из истории браузера, так чтобы кнопка браузера Назад после перемещения не вела к ней.

Решение

Для того чтобы достигнуть такого эффекта, к новой странице нужно перемещаться, используя метод location.replace():

```
location.replace("http://www.megacorp.com/indexDHTML.html");
```

Обсуждение

Возможность удалить страницу из истории переходов может стать удобной в том случае, если сайт содержит страницу, автоматически перенаправляющую браузер по другому адресу с применением сценария. В рецепте 5.10 представлена типичная ситуация, в которой эта возможность применима. Без этого пользователь, достигнув конечной страницы и щелкнув на кнопке Назад, вернулся бы обратно на временную страницу и тем самым попал бы в бесконечный цикл перемещений, выбраться из которого можно, только перемещаясь вперед.

Кроме того, данную методику можно применить для того, чтобы пользователь не мог вернуться на страницу с формой после отправки. Но в этом случае придется собирать данные из формы самостоятельно, присоединяя их в виде аргументов к URL новой страницы. Тогда страница, возвращенная CGI-программой на сервере, заместит собой текущую страницу в истории переходов. Следует учитывать, что такой метод работает только для форм, использующих метод GET. и непригоден в случае применения POST. Это связно с тем, что вызов location.replace() заставляет браузер запрашивать новую страницу, задействуя метод GET как обычную веб-страницу.

Смотрите также

Ситуация, в которой удобно применять метод location.replace(), описана в рецепте 5.10. Как собрать данные из формы и преобразовать их в часть URL, описано в рецепте 10.6.

10.3. Навигация с помощью select

NN2 **IE3**

Задача

Пользователь должен иметь возможность выбрать пункт назначения из списка в элементе select.

Решение

Существует несколько методик для получения нужного поведения, применимость которых зависит от дизайна страницы и стиля написания сценария. Но все эти методы используют элемент **select**, заполненный элементами option с информацией об URL для каждого из пунктов. Текст, который будет видеть в этом элементе пользователь, может быть любым, но в атрибут value каждой опции следует поместить URL:

```
<select name="chooser" id="chooser">
    <option value="">Выберите страницу:</option>
    <option value="http://www.megacorp.com/index.html">Домашняя</option>
    <option value="http://www.megacorp.com/products/index.html">Продукты
    </option>
    <option value="http://www.megacorp.com/support/index.html">Поддержка
    </option>
    <option value="http://www.megacorp.com/contact.html">Связь</option>
</select>
```

Для запуска процесса перехода нужно использовать какое-нибудь событие. Подходом, обладающим наибольшей обратной совместимостью, является размещение поблизости от элемента select кнопки Переход или соответствующего значка. Обработчик события onclick такой кнопки должен считывать значение, выбранное в select:

```
function navigate() {
    var choice = document.forms[0].chooser;
    var url = choice.options[choice.selectedIndex].value;
    if (url) {
        location.href = url;
    }
}
```

Но удобнее для пользователей, возможно, будет переходить на новую страницу сразу после выбора нужного значения из списка. В этом случае можно соз-

дать обобщенную (то есть пригодную к повторному использованию) функцию, которая получает в качестве одного из аргументов ссылку на элемент select:

```
function navigate(choice) {
    var url = choice.options[choice selectedIndex].value;
    if (url) {
        location.href = url;
    }
}
```

Для такой функции следует определить обработчик события следующим образом:

```
<select name="chooser" id="chooser" onchange="navigate(this)">...</select>
```

Обсуждение

Существует множество способов преобразовать строку, выбранную в select, в команду перехода. Эти способы зависят, прежде всего, от версий браузеров, которые нужно поддерживать, и стиля кодирования, особенно в части обработки событий. Например, в IE 4, Netscape 6 и старше можно получить значение выбранной опции, просто обратившись к свойству **value** элемента **select.** Такой метод проще, чем метод с использованием массива, показанный в решении. С другой стороны, если нужно связать обработчик события **onchange** с элементом **select** другими способами, обработчик события не сможет получать ссылку на элемент **select** как аргумент. В этом случае функция должна извлечь нужную информацию из объекта события способом, применимым как к модели события IE, так и NN:

```
function navigate(evt) {
   evt - (evt) ? evt : ((event) ? event : null):
   if (evt) {
      var elem = (evt.target) ? evt.target ·. ((evt.srcElement) ?
           evt srcElement : null):
      if (elem && elem.tagName.toLowerCase() -- "select" && elem.value) {
         location.href = el em.value;
      }
```

Такой обработчик события можно связать с элементом следующим образом:

```
<select name="chooser" id="chooser" onchange="navigate(event)">...</select>
```

Или же можно использовать для назначения обработчика сценарий, выполняемый после загрузки страницы:

```
document.getElementById("chooser").onchange = navigate:
```

Тем не менее в представленной схеме есть существенное упущение. Когда пользователь переходит на новую страницу с помощью select, а затем возвращается обратно, используя кнопку браузера Назад, большинство браузеров отобразит страницу точно в таком же виде, в каком она была покинута. Таким образом, в элементе select будет заранее выделен последний выбранный пользователем пункт. Суть проблемы в том, что пользователь не сможет повторить свой выбор, так

как при повторном выделении одного и того же пункта событие onchange не произойдет. Другими словами, пользователь не сможет перейти обратно, не выделив перед этим другой пункт. Можно применять для навигации одно из событий, связанное с мышью, но это рискованный путь, так как пользователь может нажимать и отпускать кнопки мыши только для того, чтобы просмотреть варианты выбора, предлагаемые в списке. Таким образом, события мыши не дают хорошей замены.

Для того чтобы сделанный пользователем выбор вызвал событие onchange, можно через события onload или onunload объекта window вызывать сценарий, восстанавливающий значение элемента select (или даже значения всех элементов управления формы) в исходное состояние. Также можно вызвать метод reset() у соответствующей формы или же установить значение свойства selectedIndex у соответствующего элемента select в 0. Следует обратить внимание на то, что в показанном решении в элементе select по умолчанию выбран пункт, которому не соответствует никакого значения, только надпись, содержащая инструкцию. Все показанные в решении обработчики события перед использованием значения проверяют, ассоциировано ли с выбранным пунктом какое-либо значение.

Может показаться непонятным, зачем разработчик сайта поместил рядом с элементом select кнопку перехода. На это есть две основные причины. Во-первых, разработчику пользовательского интерфейса может не понравиться то, что элемент select вызывает такое радикальное действие, как переход на другую страницу. Другими словами, умышленно разделяются действия выбора пункта назначения и перехода. Во-вторых, существует исторический прецедент, когда в Navigator версии 2 обработчик события onchange для элемента select работал неправильно в версии этого браузера для Windows (элемент терял фокус до того, как событие происходило). Поэтому для того, чтобы выполнить переход, разработчику приходилось добавлять рядом с элементом select кнопку или гиперссылку. Трудно сказать, насколько эти причины влияют на сегодняшнюю ситуацию. Если нужно поддерживать браузеры, не имеющие нужных возможностей, можно добавить подобную кнопку на страницу и использовать технологию проверки браузера, чтобы спрятать кнопку, если она не нужна.

Смотрите также

Управление навигацией в других фреймах показано в рецептах 7.2 и 7.3. некоторые идеи о динамическом изменении страницы по мере загрузки можно почерпнуть из рецепта 14.1.

10.4. Передача данных через cookie

NN2 **IE3**

Задача

Передать пользовательские данные с одной страницы на другую, без взаимодействия с сервером, применяя для передачи cookie.

Решение

С помощью библиотеки cookies.js, показанной в рецепте 1.9, можно в событии onunload сохранять данные пользователя в cookie, используя от одной до двадцати пар имя/значение. Следующий пример сохраняет в cookie значение, введенное пользователем в поле ввода. Сохраненное значение будет храниться на машине пользователя 180 дней:

```
<scr pt type="text/javascript" src="cookies.js"></script>
<script language="JavaScript" type="text/javascript">
function saveData') {
    var data = document.forms[0].userName.value:
    var expDate = getExpDate(180. 0. 0):
    setCookie("userName" data. expDate):
}
</script>

<body onunload="saveData()">
```

Во втором документе обработчик события onload запрашивает данные из cookie и помещает нужные значения в текстовые поля формы:

```
<script type="text/javascript" src="cookies.js"></script>
<script language="JavaScript" type="text/javascript">
function readData() {
    var data = getCookie("userName);
    document.forms[0].username value = data:
}
</script>

<body onload="readData()">
```

Можно объединить оба обработчика в каждой из страниц, передавая данные между ними во время всех посещений.

Обсуждение

В отличие от всех остальных методик передачи данных, описанных в этой главе, использование cookie позволяет сохранять данные даже после посещения сайта пользователем и выключения им компьютера. Однако если не указывать при создании cookie дату их истечения, данные исчезнут после закрытия браузера пользователем.

Данные в cookie могут содержать только строки. Таким образом, для того чтобы сохранить данные из массивов и пользовательских объектов, сформированных сценарием, нужно преобразовать их в строки (см. рецепты 3.3 и 3.13). После того как эти значения будут прочитаны, необходимо произвести обратное преобразование.

Передача данных из формы, содержащей несколько элементов управления (возможно, разных типов), требует того, чтобы каждое из значений было извлечено из формы по отдельности. Это можно делать вручную, сохраняя каждое значение в отдельный cookie, или же использовать вспомогательный сценарий

(рецепт 8.14), который извлекает данные из всех подходящих элементов управления и сохраняет их в виде строки. Параллельная функция ответственна за извлечение данных и заполнение ими идентичной формы на другой странице.

Перенос данных с помощью cookie — эффективный процесс, который полностью невидим для пользователей. Они даже не будут знать, что вы пишете и читаете cookie на их компьютере. Но пользователи могут изменить обычный механизм обработки cookie на их компьютере, что может повлиять на механизм незаметной передачи данных или даже полностью разрушить его. Например, пользователь может настроить браузер так, чтобы получать предупреждение перед каждым обращением к cookie. Но немногие пользователи выбирают такие раздражающие настройки, так как сейчас cookie очень широко используются по всей Сети. Более часто встречаются пользователи, полностью отключающие cookie. В некоторых организациях устанавливаются браузеры, в которых cookie отключены по умолчанию. Чтобы убедиться в том, что cookie доступны для сценария, можно применить методику проверки поддержки cookie (рецепт 5.9).

Следует помнить о существовании физических ограничений на количество сохраняемых в cookie данных, особенно в ранних версиях браузеров. Каждый сервер и домен может иметь не более 20 пар имя/значение. Если нужно хранить большее количество данных необходимо разработать структуру, которую можно представить в виде строки. Например, создать пользовательский объект с 25-ю свойствами и использовать рецепт 3.13 для сохранения этого объекта в строку, которую можно поместить в cookie. Существует и другое ограничение, возникающее, когда нужно сохранить большое количество данных. Наибольший обмен данных, которые можно сохранять с одного сервера и домена, ограничен 4000 символов, а количество данных, которые можно безопасно хранить в одной паре имя/значение, не превышает 2000 символов. Подсчитывая объем сохраненных данных, следует учитывать, что значения пропускаются через функцию escape(), которая расширяет некоторые символы до трех знаков, что в целом приводит к увеличению объема данных примерно на 20 %.

Смотрите также

Утилиты для работы с cookie представлены в рецепте 1.9. Сохранение массивов и преобразования строк описаны в рецепте 3 3 Определить, поддерживаются ли cookie, поможет рецепт 5.9. В рецепте 8.14 показано, как преобразовать все данные формы в строку для дальнейшей передачи.

10.5. Передача данных через фреймы

NN2 IE3

Задача

Передать пользовательские данные с одной страницы на другую, без взаимодействия с сервером, применяя для передачи фреймы.

Решение

В то время как в отдельных фреймах меняются документы, объект window, представляющий сам набор фреймов, остается неизменным. Это окно верхнего уровня можно использовать для хранения данных JavaScript любого типа в виде глобальных переменных. В следующем примере строка из текстового поля ввода сохраняется в глобальной переменной, называемой userName:

```
<script language="JavaScript" type="text/javascript" >
function saveData() {
    top.userName - document.forms[0].userName.value;
}
</script>

<body onunload="saveData()">
```

Во втором документе обработчик события onload извлекает данные из переменной и помещает их в текстовое поле:

```
<script language="JavaScript" type="text/javascript" >
function readData() {
    if (typeof top.userName != "undefined") {
        document.forms[0].userName.value = top.userName;
    }
}
</script>

<body onload="readData()">
```

Оба сценария и обработчика события можно встроить в каждую из страниц, чтобы данные перемещались между ними при любых переходах.

Обсуждение

Использование для передачи данных фрейма имеет свои преимущества, но следует помнить и о недостатках. Одно из главных преимуществ в том, что в виде переменной можно хранить данные любого типа, не преобразуя их в строки. В то же время нет ограничений на объем хранимых данных, имеющихся при использовании cookie или URL. JavaScript автоматически объявляет и инициализирует глобальную переменную при присвоении ей значения.

Значение, присвоенное переменной, хранящейся в окне или фрейме верхнего уровня, хранится все время, пока задающий структуру фреймов документ остается загруженным. Таким образом, если дизайн сайта позволят пользователям в одном из фреймов перемещаться за пределы основного сайта, значение переменных сохранится к тому времени, когда пользователь вернется назад. Сценарии на страницах с другого сервера или домена не смогут обратиться к этим переменным из за политики безопасности общего источника, применяемой при доступе к переменным в современных браузерах.

Пожалуй, одним из самых уязвимых моментов при использовании фреймов для хранения данных является то, что сохраненные данные в общем случае не

переживают перезагрузки страницы. Если пользователь случайно или намеренно щелкнет на кнопке браузера Обновить, перезагрузится сам набор фреймов и все переменные исчезнут. Вот почему показанная в примере функция readData() проверяет существование переменных перед обращением к ним. Если рассчитывать на то, что данные выдерживают изменение документа в не более чем одном фрейме, эта проблема уже не кажется столь значительной.

С другой стороны, не каждый веб-разработчик использует в своей работе фреймы, чему могут быть разные причины. Пользователей же это обычно не заботит, пока они не сталкиваются с затруднениями в навигации, связанными с применением фреймов (в частности, странное поведение кнопок Вперед и Назад). Один из путей разрешить ситуацию, не отказываясь от фреймов, — создать структуру, состоящую из двух фреймов, один из которых невидим:

```
<frameset rows="100%, *" border="0">
   <frame name="main" src="content.html">
   <frame name="hidden" src="blank.html">
</frameset>
```

Такая структура предоставляет разработчику два объекта window, которые можно использовать как хранилища для данных. окно верхнего уровня и невидимый фрейм (parent.hidden). Некоторые дизайнеры применяют скрытые от пользователя фреймы для размещения в них Java-апплетов, не имеющих интерфейса. Такие апплеты могут поддерживать соединение с сервером, передавая в реальном времени данные в основную страницу.

Схему с использованием невидимого фрейма можно улучшить, избавившись от недостатков, возникающих из-за того, что информация хранится в переменных. Многие (хотя не все) браузеры сохраняют данные в формах при перезагрузке страницы. Поэтому вместо того, чтобы сохранять данные в переменных, можно помещать их в текстовые поля или элементы area. Например, если для сайта разрабатывается корзина покупок, можно накапливать данные о заказах в полях формы на невидимом фрейме, по мере того как пользователь перемещается по страницам с информацией о товарах. Если пользователь захочет просмотреть свои заказы, страница с корзиной покупок должна запрашивать данные из невидимого фрейма и формировать на их основе красиво оформленную страницу (возможно, применяя для формировании HTML-кода оператор document.write()). При использовании такой технологии сохранения данных, особенно в приложениях электронной коммерции, следует обязательно проверить, насколько надежно данные сохраняются при перезагрузке страницы. Такую проверку необходимо провести в как можно большем количестве разных браузеров и операционных систем, желательно проверить, что произойдет, если уйти с сайта, используя кнопку Назад, а затем вернуться обратно при помощи кнопки Вперед. Если данные в скрытых текстовых полях сохранились, это хороший знак.

Смотрите также

Управление фреймами и их структурой описано в главе 7.

10.6. Передача данных через URL

NN2 **IE4(Mac)/5(Win)**

Задача

Передать данные, введенные пользователем, из одной страницы в другую, не вовлекая в передачу сервер, cookie или фреймы.

Решение

Данные можно передать через аргументы в URL открываемой страницы, содержащей сценарий, который прочитает аргументы и восстановит из них данные. В простейшем случае можно использовать следующий код, который передает единственное значение из текстового поля:

```
<script language="JavaScript" type="text/javascript">
function goNext(url) {
    var data = document.forms[0].userName.value;
    location.href = url + "?" + escape(data);
}
</script>

<a href="page3.html" onclick="goNext('page3.html'); return false;">___</a>
```

Второй документ извлекает данные из аргументов URL и помещает их в текстовое поле с тем же именем:

```
<script language="JavaScript" type="text/javascript">
function readData() {
    var srchString = unescape(location.search.substring(1,
        location.search.length));
    if (srchString.length > 0) {
        document.forms[0].userName.value = srchString;

}
</script>

<body onload="readData()">
```

Если необходимо, чтобы данные передавались между всеми страницами, можно включить оба сценария в каждую страницу.

Обсуждение

В основе этого метода лежит то, что все версии Netscape Navigator, IE 4 и старше для Macintosh и IE 5 и старше для Windows сохраняют аргументы URL-страницы, пришедшей с сервера, даже если самим сервером эта информация не используется. Строка аргументов начинается с символа-разделителя **?**, который следует за адресом страницы. Этот символ следует обязательно включать при формировании URL для следующей страницы. Соответственно, функция, извлекающая данные из URL, должна прежде всего искать этот символ.

В решении представлен очень простой случай, когда с одной страницы на другую передается единственное строковое значение. Но между страницами можно передавать и куда более сложные данные. Обычно можно преобразовать данные в набор пар имя/значение, как это делается при обычной отсылке формы, разделяя имя и значение знаком равенства, а друг от друга пары отделяются знаком амперсанда (&):

```
URLстраницы?имя1=значение1&имя2=значение2&имя3=значение3
```

Тем не менее следует учитывать, что задача передачи данных с одной страницы на другую — обеспечить возможность использовать эти данные на последующих страницах. Так как формат пар имя/значение отличается от формата, применяемого в JavaScript, перед передачей объектов и массивов JavaScript через аргументы URL нужно использовать некоторый сценарий, чтобы преобразовать их в приемлемую форму. Соответственно, на странице, принимающей данные, их необходимо преобразовать обратно в пригодную для использования форму.

Для этой цели можно взять библиотеку преобразования объектов в строки, показанную в рецепте 3.13. Как и данные, хранящиеся в cookie, передаваемые через аргументы URL данные могут (а некоторые считают, что должны) быть оформлены в виде пар имя/значение, которые можно преобразовать в объект JavaScript. Затем этот объект можно применить для заполнения элементов управления на форме или просто хранить в виде глобальной переменной, готовой для включения в сценарии. Показанный ниже фрагмент сценария демонстрирует, как можно передавать поля объекта JavaScript через аргументы URL, используя для этого средства библиотеки **objectsArraysStrings.js** из рецепта 3.12:

```
<script language="JavaScript" type="text/javascript"
src="objectsArraysStrings.js"></script>
<script language="JavaScript" type="text/javascript">
var customObject;
function goNext(url,obj) {
    srchString = object2String(obj);
    url += "?" + escape(data);
    location.href = url;
}
function readData() {
    var srchString = unescape(location.search.substring(1,
        location.search.length));
    if (srchString.length > 0) {
        customObject = string2Object(srchString);
    }
}
</script>

<body onload="readData()">

<a href="page3.html" onclick="goNext('page3.html' customObject); return false:">. .</s>
```

Если цель состоит в том, чтобы передать значения всех элементов управления некоторой формы, для упрощения сбора и распределения информации можно задействовать библиотеку **stringForms.js** из рецепта 8.14:

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

```
function goNext(url,form) {
    srchString = form2ArrayString(form):
    url += "?" + escape(data);
    location.href = url;

function applyValues(form) {
    var srchString = unescape(location.search.substring(1,
        location.search.length));
    if (srchString.length > 0) {
        string2FormObj(form, srchString):
    }
}
</script>

<body onload="applyValues(document.forms[1])">

<a href="page3.html" onclick="goNext('page3.html', document.forms[1]); return
false:">...</a>
```

Обратите внимание на то, что обе предложенные функции универсальны, так как принимают ссылку на форму, с которой ведется работа. Это позволяет при наличии на странице нескольких форм указать, какую из них следует использовать. Тем не менее не стоит обманываться кажущейся простотой показанных примеров, так как используемые здесь библиотеки решают сложные задачи по преобразованию объектов в строке и по управлению элементами форм. С другой стороны, в этом и заключается цель универсальных библиотек вспомогательных функций.

Смотрите также

Преобразования объектов в строки и обратно поможет выполнить рецепт 3.13. В рецепте 7.6 дан пример того, как можно применить передачу данных через URL, чтобы убедиться в том, что нужная страница из набора фреймов всегда загружается в этом наборе. Извлечение всех данных для передачи их в строковой форме демонстрируется в рецепте 8.14.

10.7. Создание контекстного меню

NN6 **IE5(Win)**

Задача

Необходимо вывести собственное меню по щелчку правой кнопкой мыши (в Windows) или при удержании кнопки (в Mac или Netscape), то есть в случаях, в которых обычно появляется внутреннее контекстное меню браузера.

Решение

Следует использовать обработчик события oncontextmenu, имеющийся в новых версиях браузеров, чтобы перехватить нормальную реакцию браузера и вывести собственное меню. Пример страницы, показанный в обсуждении, показывает,

как сформировать меню из стандартных элементов HTML и сценариев, которые управляют видимостью, положением и действиями каждого меню. На рис. 10.1 показан конечный результат.

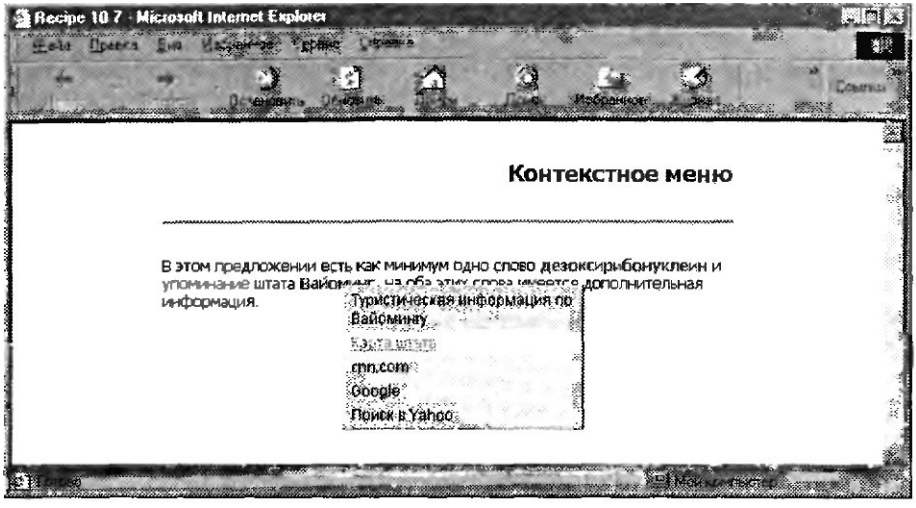


Рис. 10.1. Контекстно-зависимое всплывающее меню

Чтобы применить этот рецепт в собственных страницах, сделайте следующее:

- напишите, следуя показанной в решении модели, HTML для элементов div, представляющих собой контекстные меню;
- укажите идентификаторы для элементов span, которые заключают в себе подсвеченные слова;
- укажите данные для объекта cMenu, в частности индексы строк для отдельных пунктов меню (аналогичные идентификаторам подсвеченных span), идентификатор элемента div, содержащего меню, а также URL, куда должен вести каждый из пунктов;
- измените таблицы стилей для выделенных пунктов для лучшего соответствия стилю вашей страницы, так же как и другие таблицы стилей.

Все действия начинаются с назначения обработчиков событий мыши сразу после загрузки страницы. Показанная ниже функция initContextMenus() запускается обработчиком события onload:

```
function initContextMenus() {
    if (document.body.addEventListener) {
        // Модель события W3C DOM
        document.body.addEventListener("contextmenu", showContextMenu,
          true);
        document.body.addEventListener("click" hideContextMenus, true);
    } else {
        // Модель события IE
        document.body.oncontextmenu = showContextMenu;
    }
```

```
    // устанавливаем всплывающие подсказки
    setContextTitles();
}
```

Сценарий позволяет корректно прекращать выполнение кода в браузерах, не поддерживающих обработчик события oncontextmenu.

Обсуждение

В листинге 10.1 показан HTML-код страницы с внедренной в нее таблицей стилей. Эта страница использует контекстные меню, приводимые в действие сценарием в листинге 10.2. Страница построена вокруг основного текста, содержащего выделенные слова, для которых имеются отдельные контекстные меню. Контекстное меню можно увидеть на рис. 10.1.

Листинг 10.1. **Код HTML и CSS для контекстных меню**

```
<html>
<head>
<title>Recipe 10.7</title>
<link rel="stylesheet" id="mainStyle" href="../css/cookbook.css" type="text/css" />

<style type="text/css">
.contextMenus {position:absolute; background-color:#cfcfcf;
               border-style:solid; border-width:1px;
               border-Color:#EFEFEF #505050 #505050 #EFEFEF;
               visibility:hidden}
.menuItem {cursor:pointer; font-size:9pt;
           font-family:Arial, Helvetica, sans-serif;
           padding-left:5px; color:black;
           background-color:transparent;
           text-decoration.none}
.menuItemOn {cursor:pointer; font-size:9pt;
             font-family:Arial, Helvetica, sans-serif;
             padding-left:5px; color:red;
             background-color:yellow;
             text-decoration:underline}
.contextEntry {font-weight:bold; color:darkred; cursor:pointer}
</style>

<script type="text/javascript" src="contextMenus.js"></script>
</head>
<body onload="initContextMenus()">
<h1>Контекстное меню</h1>
<hr />

<p>В этом предложении есть как минимум одно слово <span id="lookup1"
class="contextEntry">дезоксирибонуклеин</span>
и упоминание штата <span id="lookup2" class="contextEntry">Вайоминг</span> на оба этих
слова имеется дополнительная информация.</p>

<div id="contextMenu1" class="contextMenus" onclick="hideContextMenus()"
onmouseup="execMenu(event)" onmouseover="toggleHighlight(event)"
onmouseout="toggleHighlight(event)">
```

```
<table><tbody>
<tr><td class="menuItem">Словарь Merriam-Webster</td></tr>
<tr><td class="menuItem">Тезаурус Merriam-Webster</td></tr>
</tbody></table>
</div>

<div id="contextMenu2" class="contextMenus" onclick="hideContextMenus()"
onmouseup="execMenu(event)" onmouseover="toggleHighlight(event)"
onmouseout="toggleHighlight(event)">
<table><tbody>
<tr><td class="menuItem">Туристическая информация по Вайомингу</td></tr>
<tr><td class="menuItem">Карта штата</td></tr>
<tr><td class="menuItem">cnn.com</td></tr>
<tr><td class="menuItem">Google</td></tr>
<tr><td class="menuItem">Поиск в Yahoo</td></tr>
</tbody></table>
</div>

</body>
</html>
```

Здесь приведен пример параграфа текста, содержащего несколько элементов span, окружающих выделенные слова. Имя класса, назначенное элементу span, позволяет не только управлять его внешним видом с помощью правил CSS, но и помогает отображать контекстные меню.

Два контекстных меню сделаны на скорую руку с помощью маленьких таблиц, размещенных внутри элементов **div**. Внешним видом контекстных меню, которые изначально скрыты от пользователя, управляют три правила из таблицы стилей: **contextMenus**, **menuItem** и **menuItemOn**. Каждый элемент **div** имеет обработчики для некоторых других событий, кроме отображения контекстного меню, необходимые для реакции на перемещение мыши и для перехода на нужную страницу в ответ на щелчок на пункте.

Сценарии, приводящие в действие контекстные меню, содержатся в библиотеке **contextMenu.js**, которая показана в листинге 10.2

Листинг 10.2. Библиотека contextMenu.js

```
// объекты данных для контекстных меню
var cMenu = new Object();
cMenu["lookup1"] = {menuID:"contextMenu1",
hrefs:["http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=dezoxyribonucleide" "http:/
/www.m-w.com/cgi-bin/dictionary?book=Thesaurus&va=dezoxyribonucleide"]};
cMenu["lookup2"] = {menuID:"contextMenu2", hrefs:["http://www.wyomingtourism.org/",
"http://www.pbs.org/weta/thewest/places/states/wyoming/","http://cnn.looksmart.com/r_search?
l&izch&pin=020821x36b42f8a561537f36a1&qc=&col=cnni&qm=0&st=1&nh=10&rf=1&venue=all&keyword=&q
p=&search=0&key=wyoming","http://google.com","http://search.yahoo.com"]};

// расположение и отображение контекстных меню
function showContextMenu(evt) {
    // на всякий случай прячем контекстное меню
    hideContextMenus();
    evt = (evt) ? evt : ((event) ? event : null);
    if (evt) {
```

продолжение

Листинг 10.2 (продолжение)

```
        var elem = (evt.target) ? evt.target  evt.srcElement;
         if(elem.nodeType == 3) {
            elem = elem.parentNode;
        }
        if (elem.className == "contextEntry") {
            var menu = document.getElementById(cMenu[elem.id].menuID);
            // включаем перехват событий в IE
            if (menu.setCapture) {
                menu.setCapture();
            }
            // располагаем меню по положению мыши
            var left, top;
            if (evt.pageX) {
                left = evt.pageX;
                top = evt.pageY;
            } else if (evt.offsetX || evt.offsetY) {
                left - evt.offsetX;
                top = evt.offsetY;
            } else if (evt.clientX) {
                left = evt.clientX;
                top = evt.clientY;
            }
            menu.style.left = left + "px";
            menu.style.top = top + "px";
            menu.style.visibility = "visible";
            if (evt.preventDefault) {
                evt.preventDefault();
            }
            evt.returnValue = false;
        }
    }
}

// получаем URL из объекта cMenu, соответствующего выбранному пункту
function getHref(tdElem) {
    var div = tdElem.parentNode.parentNode.parentNode.parentNode;
    var index = tdElem.parentNode.rowIndex;
    for (var i in cMenu) {
        if (cMenu[i].menuID - div.id) {
            return cMenu[i].hrefs[index];
        }
    }
    return "";
}

// переход к выбранному пункту меню
function execMenu(evt) {
    evt = (evt) ? evt : ((event) ? event  null);
    if (evt) {
        var elem = (evt.target) ? evt.target : evt.srcElement;
        if (elem.nodeType == 3) •{
            elem = elem.parentNode;
        }
        if (elem.className == "menuItemOn") {
            location.href = getHref(elem);
```

```
        )
        hideContextMenus();
    ]
}

// прячем все контекстные меню
function hideContextMenus() {
    if (document.releaseCapture) {
        // отключаем перехват событий в IE
        document.releaseCapture();
    }
    for (var i in cMenu) {
        var div = document.getElementById(cMenu[i].menuID)
        div.style.visibility = "hidden";
    }
}
// выделение цветом пунктов меню, над которыми находится мышь
function toggleHighlight(evt) {
    evt = (evt) ? evt : ((event) ? event : null);
    if (evt) {
        var elem = (evt.target) ? evt.target : evt.srcElement;
        if (elem.nodeType == 3) {
            elem = elem.parentNode;
        }
        if (elem.className.indexOf("menuItem") != -1) {
            elem.className = (evt.type == "mouseover") ? "menuItemOn"
              : "menuItem";
        }
    }
}

// установка всплывающих подсказок
function setContextTitles() {
    var cMenuReady = (document.body.addEventListener || typeof
       document.oncontextmenu != "undefined")
    var spans = document.body.getElementsByTagName("span");
    for (var i = 0; i < spans.length; i++) {
        if (spans[i].className == "contextEntry") {
            if (cMenuReady) {
                var menuAction = (navigator.userAgent.indexOf("Mac") != -1)
? "Удерживайте кнопку мыши нажатой. " : "Щелкните правой кнопкой мыши. ";
                spans[i].title = menuAction + "чтобы меню."
            } else {
                spans[i].title = "Контекстные меню доступны в других" +
                  "браузерах (IE5+/Windows, Netscape 6+)."
                spans[i].style.cursor = "default";
            }
        }
    }
}

// назначение обработчиков и инициализация всплывающих подсказок
function initContextMenus() {
    if (document.body.addEventListener) {
```

продолжение

Листинг 10.2 (продолжение)

```
        // модель события W3C DOM
        document.body.addEventListener("contextmenu" showContextMenu,
          true);
        document.body.addEventListener("click", hideContextMenus, true);
    } else {
        // модель события IE
        document.body.oncontextmenu = showContextMenu;
    }
    // Установка всплывающих подсказок
    setContextTitles();
}
```

В самом начале сценария данные для пунктов контекстных меню (в основном это URL для каждого из пунктов) помещаются в объект cMenu, при этом в качестве индексов используются идентификаторы элементов span. Остальная часть сценария разбита на три категории: назначение обработчиков событий, скрытие и отображение контекстных меню и действия в соответствии с выбранными пунктами меню.

Назначение обработчиков событий происходит в функции initContextMenus(), вызываемой из обработчика события onload тега <body>. В процессе инициализации вызывается функция setContextStrings(), которая устанавливает соответствующие операционной системе всплывающие подсказки для выделенных пунктов, используя для этого атрибут title. Отображающая контекстные меню функция, showContextMenu(), отфильтровывает только те события, которые относятся к контекстно-чувствительным элементам span. Убирает контекстные меню с экрана функция hideContextMenus(), работа которой гораздо проще, так как она скрывает все меню, и видимые и невидимые.

Когда пользователь проводит мышью над видимым контекстным меню, графические эффекты показывают пользователю, какой выбор будет сделан. Обработчики событий назначены элементам div, которые содержат таблицы и клетки таблиц, служащие источником событий. Таким образом, здесь используется механизм передачи событий.

В тот момент, когда пользователь выбирает один из пунктов меню, функция execMenu() должна определить, какой пункт из какого меню выбран, и доставить нужный URL из объекта cMenu. Ключом является вспомогательная функция getHref(), которая, получив идентификатор элемента td, в котором произошло событие mouseup, восстанавливает соответствующий URL. Эта функция вызывается из execMenu() и обрабатывает событие mouseup, переданное из элемента td (или, в NN 6 и старше, его текстового узла).

Важный стилистический момент, на который следует обратить внимание в этом рецепте, — это использование таблиц в самих контекстных меню. У такого подхода есть два основных преимущества, каждое из которых улучшает стиль и упрощает работу. Применяя внутри элемента div таблицу, не нужно беспокоиться о ширине элемента. Он самостоятельно определяет собственную ширину, основываясь на пространстве, занимаемым таблицей, которое, в свою очередь, зависит от размеров самой широкой клетки. Такого эффекта автоматической настройки ширины можно добиться и другим путем, заполнив div несколькими вложенными элемен-

тами div или p. Однако этому варианту присущ тот недостаток, что задний фон каждого из элементов имеет размер, соответствующий ширине текста. Если попытаться выделить цветом элемент, над которым пользователь провел мышью, изменяя для этого цвет заднего фона элемента, выделяемая область будет иметь разную ширину для разных пунктов меню. В таблице же ширина у всех элементов td одинакова. С другой стороны, некоторые настроены против того, чтобы использовать таблицы только для оформления. Нужных эффектов, по их мнению, следует добиваться, применяя позиционирование и таблицы стилей.

Если вы предпочитаете эту технику, можете посмотреть пример оформления всплывающих меню таблицами стилей в рецепте 10.9. Но если вы используете еще более сложные методики создания всплывающих меню, обеспечиваемые коммерческими библиотеками, то чтобы добиться нужных размеров, придется действовать методом проб и ошибок.

Хотя в данном рецепте для вызова меню служит обработчик события oncontextmenu, это не значит, что нельзя применять для этой цели другие события. Перемещение мыши (событие mousemove) подходит ничуть не хуже и работает во многих браузерах.

В процессе решения задачи возникает вопрос: как связать контекстные меню с набором соответствующих URL? В рецепте для этого служит объект (cMenu), являющийся таблицей, индексом в которой служат строки. Ради производительности (то есть мгновенной реакции на щелчок мыши), значения индексов соответствуют идентификаторам выделенных элементов span. Благодаря этому ускоряется выполнение функции showContextMenu(), которая должна определить идентификатор элемента div, соответствующего нужному контекстному меню Его можно определить так: cMenu.["spanID"].menuID.

Каждая запись, хранящаяся в объекте cMenu, сама является объектом, имеющим два свойства. Первое, menuID, хранит идентификатор контекстного меню, ассоциированного с выделенным пунктом. Второе свойство представляет массив URL, соответствующих каждому из пунктов меню. Порядок, в котором URL расположены в массиве, совпадает с порядком следования элементов td. Получая на входе выбранный пользователем элемент td, функция getHref() может вычислить ключевую информацию для извлечения URL из объекта cMenu, это идентификатор контекстного меню, которому принадлежит выбранный td и номер строки, в которой этот элемент расположен.

Существуют и другие способы реализации, более простые, чем окольный подход с использованием объекта cMenu, но они влекут за собой необходимость включать в HTML-код страницы большее количество специфических данных и в том числе применять пользовательские атрибуты. В этой книге такие методики избегаются, так как без применения пространств имен XML- и XHTML-программирования пользовательские атрибуты не проходят проверку соответствия стандартам. Конечно, не всех разработчиков волнует строгое соответствие страницы стандартам. В этом случае можно упростить работу функции execMenu(), добавив в элемент td атрибут, указывающий на связанный с ним URL. Например, если в элементе td имеется атрибут href, содержащий URL для пункта меню, то функции execMenu() вообще не нужно вызывать getHref(), достаточно просто прочитать значение URL из свойства:

```
location.href = elem.getAttribute("href");
```

Жизненно важным аспектом этого рецепта является обработка событий. Приведенный здесь код успешно работает с методиками перехвата событий как в IE, так и в W3C DOM, которые имеют мало общего. В браузере, использующем модель W3C DOM, функция initContextMenus() устанавливает обработчики событий oncontextmenu (это событие поддерживается NN 6 и старше, хотя и не является официально событием W3C DOM Level 2) и onclick в фазу перехвата событий. Это позволяет обработать события до того, как они достигнут своих целей, и тем самым предотвратить обычное срабатывание событий, когда это нужно.

В IE перехват событий используется только для событий мыши и рассматривается как временное средство. В режиме перехвата все события перенаправляются в элемент, для которого вызван метод setCapture(). При этом браузер переходит в своего рода модальное состояние, когда пользователь не может обратиться к другим элементам страницы, так как события из них автоматически передаются в вызвавший метод элемент. Так, каждому элементу div назначен обработчик событий onclick, который скрывает контекстные меню. Это нужно для того, чтобы убирать с экрана меню, если пользователь щелкнет мышкой где-либо вне его, имитируя поведение обычных контекстных меню браузера. Кроме того, когда контекстное меню скрывается, отключается перехват событий и действия мыши начинают обрабатываться обычным образом.

Смотрите также

Позиционирование элементов на странице показывается в рецепте 13.1. Управление видимостью элементов смотрите в рецепте 12.7.

10.8. Раскрывающиеся меню

NN6 IE5

Задача

Нужно, чтобы меню раскрывались из строки с заголовками на странице.

Решение

В этом решении показан один из дюжины способов реализации выпадающих меню. Рецепт использует несколько простых изображений для видимых постоянно заголовков меню, несложную внешнюю таблицу стилей, названную menus.css (листинг 10.3), и библиотеку JavaScript menus.js (листинг 10.4). Результат можно увидеть на рис. 10.2.

Чтобы применить это решение, нужно нарисовать (или позаимствовать где-нибудь) нормальные и подсвеченные изображения заголовков меню. Названия пунктов меню и соответствующие им URL нужно вписать в библиотеку menu.js. Эта библиотека собирает DHTML-компоненты для меню под управлением таблицы стилей menus.css.

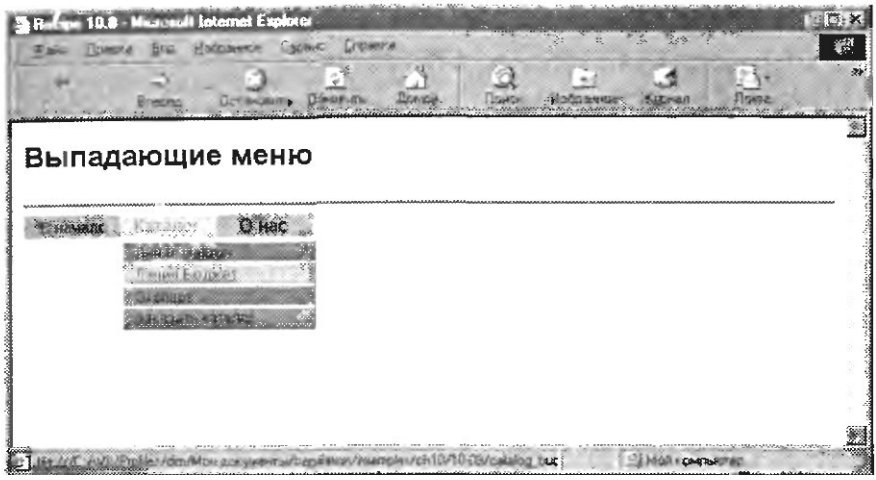


Рис. 10.2. Действие раскрывающихся меню

Обсуждение

Панель меню встроена в HTML-код страницы в виде элемента div, содержащего три изображения. Каждое из изображений окружено гиперссылкой, вызывающей основные действия для управления меню. Обработчики событий мыши назначаются из сценария:

```
<div id="menubar">
<a href="index.html"><img id="menuImg_1" class="menuImg"
src="home_off.jpg" border="0" height="20"
width="80"></a><a href="catalog.html"><img id="menuImg_2"
class="menuImg" src="catalog_off.jpg" border="0" height "20"
width="80"></a><a href="about.html"><img id="menuImg_3"
class="menuImg" src="about_off.jpg" border="0"
height="20" width="80"></a>
</div>
```

Используемая на этой странице таблица стилей, **menus.css**, содержит описание как контейнера для меню (класс **menuWrapper**, назначаемый сценарием при инициализации), так и отдельных пунктов меню, в обычном и выделенном состоянии. Эта таблица стилей показана в листинге 10.3.

Листинг 10.3. Таблица стилей menus.css

```
.menuWrapper {
            position:absolute;
            width:162px;
            background-color:#339966;
            visibility:hidden;
            border-style:solid;
            border-width:2px;
            border-color:#EFEFEF #505050 #505050 #EFEFEF;
            display:block;
            padding:3px;
```

продолжение

Листинг 10.3 (продолжение)

```
.menuItem    (
             cursor:pointer;
             font-size:12px;
             font-family:Arial,Helvetica,sans-serif;
             border-bottom:1px solid #505050;
             border-top:1px solid #EFEFEF;
             padding-left:10px;
             color:black;
             background-color:#339966;
             text-decoration:none;
             position:absolute;
             left:0px;
             width:159px;
             height:1.4em;
             display:block;
             line-height:1.4em
            1
.menuItemOn  {
             cursor:pointer;
             font-size:12px;
             font-family:Arial, Helvetica, sans-serif;
             border-bottom:1px solid #505050;
             border-top:1px solid #EFEFEF;
             padding-left:10px;
             color:#000000;
             background-color:#33ff00;
             text-decoration:underline;
             position:absolute;
             left:0px;
             width:159px;
             height:1.4em;
             display:block;
             line-height:1.4em
            }
```

Все сценарии, используемые для работы с меню, содержатся в библиотеке menus.js, показанной в листинге 10.4.

Листинг 10.4. Библиотека выпадающих меню menus.js

```
// Признак готовности меню
var menuReady = false;

// Предварительная загрузка изображения для строки меню
if (document.images) {
    var imagesNormal = new Array();
    imagesNormal["home"] = new Image(20, 80);
    imagesNormal["home"].src = "home_off.jpg";
    imagesNormal["catalog"] = new Image(20, 80);
    imagesNormal["catalog"].src = "catalog_off.jpg";
    imagesNormal["about"] = new Image(20, 80);
    imagesNormal["about"].src = "about_off.jpg";

    var imagesHilite = new ArrayO;
    imagesHilite["home"] = new Image(20, 80);
```

```
    imagesHilite["home"] src = "home_on.jpg":
    imagesHilite["catalog"] = new Image(20, 80):
    imagesHilite["catalog"].src  - "catalog_on.jpg":
    imagesHilite["about"] = new Image(20. 80):
    imagesHilite["about"].src  - "about_on.jpg":
}

function getElementStyle(elem. IEStyleProp. CSSStyleProp) {
    if (elem.currentStyle) {
        return elem.currentStyle[IEStyleProp]:
    } else if (window.getComputedStyle) {
        var compStyle = window.getComputedStyle(elem. ""):
        return compStyle.getPropertyValue(CSSStyleProp);
    }
    return "":
}

// переносим значения некоторых важных CSS-атрибутов
var CSSRuleValues = {menuItemHeight:"18px".
                     menuItemLineHeight:"1.4em".
                     menuWrapperBorderWidth:"2px".
                     menuWrapperPadding:"3px".
                     defaultBodyFontSize:" 12px"
                    }:

// specifications for menu contents and menubar image associations
var menus = new Array();
menus[0] = {mBarImgId:"menuImg_1".
            mBarImgNormal:imagesNormal["home"],
            mBarImgHilite:imagesHilite["home"],
            menuItems:[].
            elemId:""
           }:
menus[1] = {mBarImgId:"menuImg_2".
            mBarImgNormal:imagesNormal["catalog"],
            mBarImgHilite:imagesHilite["catalog"],
            menuItems:[ {text:"Линия Deluxe". href:"catalog_deluxe.html"}.
                        {text:"Линия Бюджет". href:"catalog_budget.html"}.
                        (text:"Экспорт". href:"catalog_export.html"}.
                        (text:"Заказать каталог". href:"catalog_order.html"}
                      ].
            elemId:""
           }:
menus[2] = {mBarImgId:"menuImg_3".
            mBarImgNormal:imagesNormal["about"],
            mBarImgHilite:imagesHilite["about"],
            menuItems:[ {text:"Пресс-релиз". href:"press.html"}.
                        {text:"Руководство". href:"staff.html"}.
                        {text:"Наши офисы". href:"map.html"}.
                        {text:"История компании". href:"history.html"}.
                        {text:"Рабочие записки". href:"jobs.html"}.
                        {text:"Связь". href:"contact.html"}
                      ].
            elemId:""
```

продолжение ⇩

Листинг 10.4 (продолжение)

```
// формируем хэш-таблицу для поиска объектов меню по строковым
// идентификаторам
function makeHashes() {
    for (var 1 = 0: i < menus.length: i++) {
        menus[menus[i].elemId] = menus[i]:
        menus[menus[i].mBarImgId] = menus[ı];
    }
}

// устанавливаем обработчики событий для заголовков меню
function assignLabelEvents() {
    var elem:
    for (var i = 0: i < menus.length: i++) {
        elem = document.getElementById(menus[i].mBarImgId):
        elem.onmouseover = swap;
        elem.onmouseout = swap:
    }
}

// Эта функция, вызываемая из init(), формирует элементы div для меню и
// заполняет их содержимым  Для пользователя все эти действия незаметны
function makeMenus() {
    var menuDiv. menuItem. itemLink. mbarImg, textNode, offsetLeft.
       offsetTop:
    // определяем ключевые факторы, задающие высоту div для меню

    var menuItemH = 0:
    var bodyFontSize = parseInt(getElementStyle(document.Dody. "fontSize".
       "font-size")):
    // проверяем, не был ли размер шрифта в браузере изменен пользователем
    if (bodyFontSize — parseInt(CSSRuleValues.defaultBodyFontSize)) {
        menuItemH - (parseFloat(CSSRuleValues.menuItemHeight));
    } else {
        // неплохо работает в Netscape 7
        menuItemH - parseInt(parseFloat(CSSRuleValues.menuItemLineHeight) *
                bodyFontSize):
    }
    var heightAdjust - parseInt(CSSRuleValues.menuWrapperPaddıng) +
        parseInt(CSSRuleValues.menuWrapperBorderWidth);
    if (navigator.appName == "Microsoft Internet Explorer" &&
        navigator.userAgent.indexOf("Win") !- -1 &&
        (typeof document compatMode — "undefined" ||
        document.compatMode — "BackCompat")) {
        heightAdjust - -heightAdjust:
    }

    // используем для управления циклом создания меню массив menus
    for (var i - 0: i < menus.length; i++) {
        menuDiv - document.createElement("div");
        menuDiv.id - "popupmenu" + i:
        // сохраняем идентификатор меню в свойство элемента массива menus
        menus[i].elemId - "popupmenu" + i:
        menuDiv.className - "menuWrapper":
```

```
        if (menus[i].menuItems.length > 0) {
            menuDiv.style.height = (menuItemH * menus[i].menuItems.length) -
            heightAdjust + "px";
        } else {
           // если меню не содержит пунктов, не отображаем его
            menuDiv.style.display = "none";
        }
        // устанавливаем обработчики событий
        menuDiv.onmouseover = keepMenu;
        menuDiv.onmouseout = requestHide;

        // устанавливаем порядок слоев на тот случай, если на странице их
        // несколько
        menuDiv.style.zIndex = 1000;

        // формируем пункты меню для заполнения div
        for (var j = 0; j < menus[i].menuItems.length; j++) {
            menuItem = document.createElement("div");
            menuItem.id = "popupmenuItem_" + i + "_" + j;
            menuItem.className = "menuItem";
            menuItem.onmouseover = toggleHighlight;
            menuItem.onmouseout = toggleHighlight;
            menuItem.onclick = hideMenus;
            menuItem.style.top = menuItemH * j + "px";
            itemLink = document.createElement("a");
            itemLink.href = menus[i].menuItems[j].href;
            itemLink.className = "menuItem";
            itemLink.onmouseover = toggleHighlight;
            itemLink.onmouseout = toggleHighlight;
            textNode = document.createTextNode(menus[i].menuItems[j].text);
            itemLink.appendChild(textNode);
            menuItem.appendChild(itemLink);
            menuDiv.appendChild(menuItem);
        }
        // дописываем в документ каждый из элементов div
        document.body.appendChild(menuDiv);
    }
    makeHashes();
    assignLabelEvents();
    // предварительное позиционирование меню
    for (i = 0; i < menus.length; i++) {
        positionMenu(menus[i].elemId);
    }
    menuReady = true;
}

// инициализируем глобальную переменную, которая помогает
// управлять скрытием меню с экрана
var timer;

// Эта функция вызывается из mouseover в меню, чтобы меню не
// скрывалось событием mouseout от изображения заголовка меню
function keepMenu() {
    clearTimeout(timer);
```

продолжение

Листинг 10.4 (продолжение)

```
function cancelAll() {
    keepMenu();
    menuReady = false;
]

// Эта функция вызывается событием mouseout и приводит к сокрытию
// всех меню через 1/4 секунды, если ее не отменить.
function requestHide() {
    timer = setTimeout("hideMenus()", 250);
}

// прячем все меню и приводим изображения на панели в нормальный вид
function hideMenus() {
    for (var i = 0; i < menus.length; i++) {
       document.getElementById(menus[i].mBarImgId).src =
           menus[i].mBarImgNormal.src;
       var menu = document.getElementById(menus[i].elemId)
       menu.style.visibility = "hidden";
    }
]

// перед отображением меню позиционирует его
function positionMenu(menuId){
   // позиционируем меню относительно изображения на панели заголовков меню
    var mBarImg = document.getElementById(menus[menuId].mBarImgId);
    var offsetTrail = mBarImg;
    var offsetLeft = 0;
    var offsetTop = 0;
    while (offsetTrail) {
        offsetLeft += offsetTrail.offsetLeft;
        offsetTop += offsetTrail.offsetTop;
        offsetTrail = offsetTrail.offsetParent;
    }
    if (navigator.userAgent.indexOf("Mac") != -1 &&
        typeof document.body.leftMargin != "undefined") {
        offsetLeft += document.body.leftMargin;
        offsetTop += document.body.topMargin;
    ]
    var menuDiv = document.getElementById(menuId);
    menuDiv.style.left = offsetLeft + "px";
    menuDiv.style.top = offsetTop + mBarImg.height + "px";
}

// отображаем отдельный элемент div, содержащий меню
function showMenu(menuId) {
    if (menuReady) {
        keepMenu();
        hideMenus();
        positionMenu(menuId);
        var menu = document.getElementById(menuId);
        menu.style.visibility = "visible";
    }
}
```

```
// подмена изображений на панели меню для создания эффекта
// выделения пунктов
function toggleHighlight(evt) {
    evt = (evt) ? evt    ((event) ? event : null):
    if (typeof menuReady != "undefined") {
        if (menuReady && evt) {
            var elem = (evt.target) ? evt.target : evt.srcElement:
            if (elem.nodeType == 3) {
                elem = elem.parentNode:
            }
            if (evt.type == "mouseover") {
                keepMenu():
                elem.className ="menuItemOn":
            } else {
                elem.className ="menuItem":
                requestHideO:
            }
            evt.cancel Bubble = true:
        ]
    }
}

function swap(evt) {
    evt = (evt) ? evt : ((event) ? event . null):
    if (typeof menuReady != "undefined") {
        if (evt && (document.getElementById && document.styleSheets) &&
          menuReady) {
            var elem = (evt.target) ? evt.target : evt.srcElement:
            if (elem.className = "menuImg") {
                if (evt.type == "mouseover") {
                    showMenu(menus[elem.id].elemId);
                    elem.src = menus[elem.id].mBarImgHilite.src:
                ) else if (evt.type == "mouseout") {
                    requestHideO:
                }
                evt.cancel Bubble = true:
            ]
        }
    }
}

// создание меню, но только если поддерживаются ключевые возможности
function initMenus() {
    if (document.getElementById && document.styleSheets) {
        setTimeout("makeMenus()",5):
        window.onunload=cancelAll:
    }
```

Сценарий начинается с объявления одной глобальной переменной, menuReady, к которой различные функции обращаются, чтобы определить, когда система меню готова к анимации. Далее следует код для предварительной загрузки изображений заголовков меню в обоих (обычном и выделенном) состояниях (см. рецепт 12.1). Вспомогательная функция getElementStyle() служит для того, чтобы поддерживать размер пунктов меню в соответствии с пользовательскими настройками в браузерах, основанных на Mozilla. Эта функция основана на рецепте 11.12.

Так как ограничения, связанные с безопасностью, не позволяют сценариям считывать из таблиц стилей значение свойства rule, в сценарий включен глобальный объект, содержащий некоторые ключевые сведения, которые сценарии могут использовать для управления позиционированием и размерами меню. Эти значения берутся из таблицы стилей.

Далее следует создание объектов (помещаемых в массив menus), содержащих жизненно необходимую информацию, которая потребуется позднее, при формировании меню. Каждый объект имеет пять свойств, которые будут подробно описаны позднее.

Сценариям, ответственным за скрытие и отображение отдельных меню, зачастую нужно по ссылке на меню восстановить ссылку на изображение, служащее заголовком меню, и обратно. Чтобы ускорить этот процесс (то есть, чтобы избежать перебора всех элементов массива menu в поисках подходящего), во время инициализации создается имитированная хэш-таблица (функция makeHashes()), строковые индексы которой состоят из идентификаторов изображений и элементов меню (см. рецепт 3.9). В результате появляется хэш-таблица, в которой на каждый элемент массива menus есть две ссылки: одна адресуется идентификатором изображения, а другая — идентификатором меню. В процессе инициализации вызывается еще одна функция. assignLabelEvents(), которая устанавливает обработчики событий для изображений заголовков на панели меню.

Самая большая функция в этом рецепте, makeMenus(), формирует элементы меню, используя для их создания синтаксис W3C DOM. Эта функция также вызывается в процессе инициализации и для заполнения отдельных меню пунктами и соответствующими им URL использует массив menus, объявленный ранее.

Обработка движений мыши в этом приложении производится для управления внешним видом заголовков меню и для выделения отдельных пунктов в меню. Хотя это разные действия, они связаны друг с другом. Но перед тем, как переходить к ним, нужно сделать кое-какую грязную работу. Так как изображения на панели меню и соответствующие им меню взаимосвязаны, то ненужные более меню нужно приводить в порядок, этим процессом управляет таймер setTimeout(). Идентификатор таймера (значение, возвращаемое функцией setTimeout()) сохраняется в глобальной переменной, названной timer.

Для того чтобы убирать с экрана меню, нельзя просто использовать событие mouseout у изображения в панели меню. Если пользователь перемещает мышь с изображения заголовка меню в соответствующее меню, то убирать его с экрана не нужно. Добиться нужного поведения помогает функция keepMenu(), которая отменяет срабатывание таймера, вызывающего функцию requestHide(). Благодаря этому меню остается видимым.

Родственная функция cancelAll(), вызываемая обработчиком события onunload, помогает бороться с проблемами, которые могут возникать при загрузке новой страницы, если курсор все еще перемещается над изменяющимся изображением. Причина проблем в том, что глобальные переменные, действительные в начале функции, могут исчезнуть к тому времени, когда они понадобятся. Функция cancelAll() прекращает вычислительные процессы в странице на время перехода.

В функции requestHide(), обработчике события mouseout для панели меню и для самих меню взводится таймер, через 1/4 секунды вызывающий функцию

hideMenus(), полностью восстанавливающую исходный вид отдельных меню и панели заголовков. Таким образом, если таймер не будет отключен в течение четверти секунды, меню будут скрыты с экрана.

Перед тем как отобразить меню на экране, вызывается отдельная функция, служащая для позиционирования, positionMenu(). Благодаря ей меню будет выведено в нужном месте, даже если панель меню изменила свое положение на странице (например, вследствие изменения размера окна браузера).

Изображения на панели меню и отдельные компоненты самих меню при получении события mouseover вызывают функцию showMenu(), отключающую тикающий таймер, тем самым отменяющую вызов hideMenu(). После того как таймер отключен, обработчик сразу же прячет все меню (вместо того чтобы ждать, когда сработает таймер). После чего находит в массиве menus идентификатор (хранящийся в свойстве elemId) элемента div, содержащего нужное меню, и отображает этот элемент.

По мере того как пользователь перемещает указатель мыши над пунктами меню, обработчики событий mouseover и mouseout меняют оформление элементов (для этого в функции toggleHighlight() меняется CSS-класс элемента). Кроме того, обработчик события mouseover вызывает keepMenu(), чтобы удостовериться, что таймер, убирающий с экрана меню, отключен. При этом если произошло событие mouseout, таймер не отменяется и меню скрываются с экрана (если не произошло новых mouseover).

Функция swap(), вызываемая из обработчиков событий мыши в панели меню, работает как переключатель, отображающий или скрывающий выпадающие меню. В ответ на событие mouseover отображается нужное выпадающее меню, а изображение на панели меню подменяется на подсвеченную версию. В ответ на mouseout вызывается requestHide(), и если таймер не будет отменен раньше, чем через 1/4 секунды, меню исчезнет, а изображение на панели меню вернется в исходное состояние.

Обработчик события onload, функция initMenus(), проверяет, имеется ли в браузере все необходимое для работы системы выпадающих меню. Она проверяет не только поддержку основных возможностей W3C DOM, но и малоизвестного свойства document.styleSheets (которое не поддерживается в Opera вплоть до версии 6). Если все необходимое в браузере имеется, функция формирует элементы div, составляющие меню. Кроме того, она назначает окну обработчик события onunload, который все запросы на закрытие меню (таймеры) отменил перед переходом на новую страницу.

Одной из основных задач, решаемых в предложной системе меню, является уменьшение количества работы, необходимой, чтобы приспособить внешний вид меню для любых приложений. Большинство доступных из других источников библиотек для работы с меню идут еще дальше в этом направлении, создавая очень сложные системы из объектов и динамически формируемого HTML. Многоцелевые библиотеки, особенно те из них, которые поддерживают устаревшие объектные модели (например, Navigator 4 DOM), вынуждены усложнять код, чтобы соответствовать как можно большему диапазону применений. С другой стороны, этот пример имеет меньший размер, из-за чего его внедрение требует некоторой дополнительной работы (в частности, разработка изображений для

панели меню). Но зато тут есть множество идей, которые можно использовать при создании более автоматизированных систем меню, совместимых с IE 5 и NN 6 и более поздними версиями

Использование для управления внешним видом меню таблиц стилей упрощает эксперименты с такими параметрами, как комбинации цветов, шрифтов и размеров. В сценарии JavaScript, полагающемся на поддержку таблиц стилей, делается несколько предположений.

Все выпадающие меню имеют одинаковую ширину, не зависящую от ширины изображений на панели меню (которые могут быть любого размера).

Ширина отдельных пунктов меню определяется путем вычитания ширины отступа в обрамляющего меню элемента из ширины этого элемента (в этом рецепте обрамляющий элемент имеет ширину 162 пиксела, а пункты меню — 159).

Таблицы стилей для двух разных состояний пунктов меню должны указывать одинаковые размеры и величину шрифта.

Некоторые свойства из таблицы стилей, такие как высота пунктов меню и ширина рамки обрамляющего элемента, должны быть повторены в объекте CSSRulesValues, объявленном в коде сценария. Если вы меняете значения в таблице стилей, вы должны внести соответствующие изменения и в код.

Чтобы привести систему меню в соответствие с вашими графическими меню, нужно сделать несколько изменений в коде. Прежде всего нужно настроить предварительную загрузку изображений в соответствии с теми рисунками, которые будут использоваться в панели меню (панель может быть и вертикальной). Строковые индексы массива изображений и соответствующие свойства src устанавливаются точно так же, как это делается в рецепте 12.1.

На следующем шаге нужно описать содержимое меню. Массив menus состоит из объектов, описывающих отдельные меню. Каждый объект имеет множество свойств, используемых на разных этапах создания, отображения и скрытия меню с экрана:

`mBarImgId`

Строка, содержащая идентификатор элемента img, соответствующего заголовку данного меню на панели меню.

`mBarImgNormal`

Ссылка на предварительно загруженное изображение, соответствующее нормальному состоянию заголовка меню.

`mBarImgHilite`

Ссылка на предварительно загруженное изображение, соответствующее подсвеченному заголовку меню.

`menuItems`

Массив объектов, по одному на каждый пункт меню (если выпадающего меню не должно быть, в этом свойстве должен храниться пустой массив). Каждый из объектов содержит два свойства, одно из них — это название пункта, а другое — URL, на который должен перейти пользователь, выбрав пункт.

`elemId`

Изначально в это свойство помещается пустая строка, а значение помещается в него при создании меню. С этим свойством делать ничего не надо.

Последнее, что нужно настроить, — это панель заголовков меню. Каждому заголовку должно соответствовать отдельное изображение (которое имеет два состояния: нормальное и подсвеченное). В HTML-коде нужно назначить обработчики событий onmouseover и onmouseout для каждого изображения. Оба обработчика должны вызывать один и тот же метод swap(), передавая ему единственный аргумент, отсчитываемый с нуля индекс элемента в массиве menus, несущего описание меню для данного заголовка.

Хотя в данной системе меню это и необязательно, рекомендуется окружать каждое изображение на панели заголовков гиперссылкой. Это позволит устаревшим браузерам и поисковым машинам перейти к другим разделам сайта, даже если странички, к которым ведут эти ссылки, содержат просто список ссылок, аналогичный содержимому меню. Те пользователи, которые могут использовать меню, не будут попадать на эти страницы.

Остальной код в рецепте ответственен за формирование меню и управление их отображением. Пункты меню содержат традиционные гиперссылки для тех пользователей, которые хотят видеть пункт назначения в строке состояния.

Код функции makeMenus() построен в расчете на то, что разрабатываемые меню являются выпадающими меню в истинном смысле этого слова, и располагает меню непосредственно под заголовком. Если нужно, чтобы меню всплывали справа от заголовка или вверх, нужно внести изменения в операторы, изменяющие значения свойств menuDiv.style.left и menuDiv.style.top. Обратите внимание на то, что в этом рецепте левая граница меню выстроена по значению свойства mBarImg.offsetLeft, а верх меню прижат к нижней части заголовка, для этого используется свойство mBarImg.height. Если нужно сделать меню, которое всплывало бы справа от заголовка, то верх меню должен иметь координату mBarImg.offsetTop, а левая часть должна быть сдвинута вправо на величину mBarImg.width. Чтобы меню появлялось над заголовком, его верхняя граница должна быть равна mBarImg.offsetTop минус высота элемента menuDiv (заданная ранее в этой функции). Во всех этих случаях нужно оставлять код, подстраивающий значения переменных offsetTop и offsetLeft, так как этот код заботится о некоторых нестандартных особенностях позиционирования элементов в Internet Explorer.

К несчастью, неуместные ограничения, наложенные на сценарии в целях безопасности, не позволяют напрямую считывать атрибуты таблиц стилей. Чтобы обойти запрет, приходится повторять значения в коде сценария (в глобальной переменной CSSRulesValues), чтобы к ним можно было обращаться при задании размеров меню. Если в будущих браузерах появится возможность обращаться к правилам таблиц стилей, не нарушая ограничений безопасности, можно будет управлять расположением и отображением меню, используя только таблицы стилей. Для будущего использования далее приведена функция, строго следующая W3C DOM (и синтаксическим особенностям IE), которая считывает значения из таблицы стилей, встроенной в HTML посредством тега style или присоединенной к документу с помощью элемента link. Сейчас функция работает в том

виде, как она есть, если запускать ее с локального жесткого диска. При обращении к странице на сервере произойдет ошибка нарушения доступа:

```
// вспомогательная функция, вызываемая из makeMenus().
// возвращающая значения атрибутов правил из таблицы стилей.
// параметры: идентификатор элемента <style>. имя селектора и имя атрибута
function getCSSRuleValue(styleID, selector, attr) {
    var sheet, styleElem, i;
    for (I = 0; i<documents.styleSheets.length; i++) {
        sheet = document.styleSheets[i];
        styleElem - (sheet.ownerName) ? sheet.ownerNode
            ((sheet.owningElement) ? sheet.owningElement  null);
        if (styleElem) {
            if (styleElem.id – styleID) {
                break;
            }
    ]
    var rules = (sheet.cssRules) ? sheet.sccRules : ((sheet.rules) ?
        sheet.rules : null);
    if (rules) {
        for (i = 0; i < rules.length; i++) {
            if (rules[i].selectorText == selector || rules[i].selectorText
                == "*" + selector) {
                return rules[i].style[attr];
            ]
        }
    }
    return null;
]
```

Если вы рассчитываете на то, что панель меню находится в одном фрейме, а меню должны появляться в другом, то придется проделать некоторую дополнительную работу. Дело в том, что элементы div существуют только в пределах одного окна браузера и поэтому не могут занимать несколько фреймов. Для начала нужно встроить в каждую из страниц во втором фрейме код, формирующий элементы div. Но этот код нужно изменить, чтобы он искал заголовки меню в другом фрейме. Кроме того, нужно учесть возможность прокрутки изменяемого фрейма, которая может изменить положение меню, несмотря на то, что положение заголовков в другом фрейме осталось неизменным. Дополнительное взаимодействие между фреймами необходимо для синхронизации изображений заголовков с отображением и скрытием меню.

Напоследок, вас может заинтересовать причина, по которой для скрытия меню используется функция **requestHide()** и **setTimeout()**. Обратите внимание на то, что меняется состояние двух разных элементов: у элемента **img** меняется атрибут src, а у меню — видимость. Обработчик события **onmouseout** должен работать только в том случае, если пользователь переместил указатель куда угодно, но не вниз, в меню. Если же курсор перемещен в меню, ни скрывать его, ни восстанавливать исходный вид заголовка не нужно. Поэтому обработчик события onmouseout с помощью setTimer() запускает таймер, который должен восстановить вид меню через 250 миллисекунд после срабатывания события. В то же время обработ-

чики событий onmouseover в элементах меню вызываются до того, как истекут 250 миллисекунд, и сбрасывают таймер, поэтому состояние элементов не меняется, пока указатель находится в меню (или на панели заголовков).

Когда пользователь перемещает курсор на заголовок другого меню, таймер тоже сбрасывается, чтобы функция hideMenus() могла скрыть все меню немедленно (благодаря этому задержка получается меньше 250 миллисекунд). У таймера есть шанс вызвать hideMenus() только в том случае, если пользователь поместил указатель куда-либо вне панели заголовков или видимого меню.

Используя для установки таймера функцию requestHide(), нужно не забывать отменять срабатывание таймера в обработчике события onunload самой страницы, используя функцию cancelHide(). Если этого не сделать, срабатывание таймера может произойти уже после того, как страница ушла, забрав с собой и сценарий. Это приведет к ошибке сценария. Если ваша страница устанавливает новый обработчик события onunload, используя для этого сценарий или другими способами, нужно позаботиться, чтобы новый обработчик вызывал cancelHide() помимо прочих своих действий.

Смотрите также

Имитация хэш-таблицы показана в рецепте 3.9. Рецепт 12.1 показывает, как предварительно загружать изображения. Управление видимостью элементов демонстрируется в рецепте 12.7. Как создать позиционируемый элемент, показывается в рецепте 13.1.

10.9. Меню, отслеживающее перемещения

NN4 IE4

Задача

На каждой странице глубоко структурированного сайта нужно поместить информацию о том, в каком месте иерархии сайта находится пользователь.

Решение

Чтобы отображать на страницах положение пользователя, можно использовать библиотеку trail.js (листинг 10.5), которую нужно адаптировать к вашему сайту. На рис. 10.3 показан пример страницы, содержащей следующий HTML-код и сценарий, вызывающий библиотечную функцию makeTrailMenu():

```
<div id="trailmenu" style="position:absolute; left:20Gpx; top:2G0px">
<script language="JavaScript" type="text/javascript">
document.write(makeTrailMenu());
</script>
</div>
```

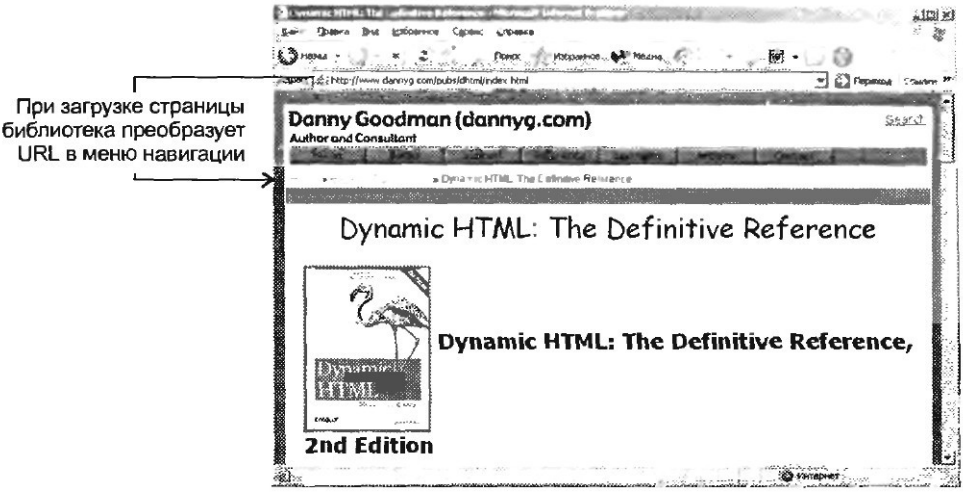


Рис. 10.3. Меню, отслеживающее положение пользователя

Обсуждение

Работа рецепта основывается на предположении о том, что структура директо рий, в которых хранятся документы, соответствует концептуальной организации сайта. Например:

```
index.html (основная страница в корневой директории)
   catalog/
      index.html (предисловие к каталогу и ссылки на отдельные категории)
      economy/
         [множество .html страниц в этой категории]
      deluxe/
         [множество .html страниц в этой категории]
      export/
         [множество .html страниц в этой категории]
   support
      index.html (страница о поддержке, содержит ссылки на категории)
      contact.html
      faq/
      downloads/
      manuals/
```

В листинге 10.5 приведен код библиотеки trail.js, которая должна быть присоединена к каждой странице, отслеживающей положение.

Листинг 10.5. Библиотека trail.js

```
var trailMenu = new Object();
trailMenu["catalog"] = "Линии продукции";
trailMenu["economy"] = "Бюджетная";
trailMenu["deluxe"] = "Люкс";
trailMenu["export"] = "Export Only";
trailMenu["support"] = "Поддержка";
```

```
trailMenu["faq"] = "FAQ (частые вопросы)";
trailMenu["downloads"] = "Файлы";
trailMenu["manuals"] = "Инструкции";

function makeTrailMenu() {
    var parseStart, volDelim, parseEnd;
    var output = "<span style='font-family:Arial, Helvetica, sans-serif; font-size:12px;
color:#000000; padding:4px'>";
    var linkStyle = "color:#339966";
    var path = location.pathname;
    var separator = " &raquo; ";
    var re = /\\/g;
    path = path.replace(re, "/");
    var trail = location.protocol + "//" + location.hostname;
    var leaves = path.split("/");
    if (location.protocol.indexOf("file") != -1) {
        parseStart = 1;
        volDelim = "/";
    } else {
        parseStart = 0;
        volDelim = "";
    }
    if (leaves[leaves.length-1] == "" || leaves[leaves.length-1] == "index.html" ||
leaves[leaves.length-1] == "default.html") {
        parseEnd = leaves.length -1;
    } else {
        parseEnd = leaves.length;
    }
    for (var i = parseStart; i < parseEnd; i++) {
        if (i == parseStart) {
            trail += "/" + leaves[i] + volDelim;
            output += "<a href='" + trail + "' style='" + linkStyle + "'>";
            output += "Home";
        } else if (i ==parseEnd - 1) {
            output += document.title;
            separator = "";
        } else {
            trail += leaves[i] + "/"
            output += "<a href-'" +trail + "' style='" + linkStyle + "'>";
            output += trailMenu[leaves[i]];
        }
        output += "</a>" + separator;
    }
    output += "</span>";
    return output;
}
```

Библиотека начинается с определения объекта, названия свойств которого соответствуют именам директорий сайта (свойства можно задавать в любом порядке). Значения свойств должны содержать понятные пользователю названия директорий. Далее следует функция makeTrailMenus(), формирующая HTML-код для навигационного меню, основываясь на пути к текущему документу и его названии. Символ "/" используется в этом примере в качестве разделителя между уровнями иерархии.

Система отслеживающих положение меню лучше всего работает на сайтах, доставляющих страницы из файлов .html, так как свойство location.pathname содержит много информации, используемой в коде для генерации меню. Эту же систему можно применить и на сайтах, формируемых серверными программами (файлы .pl, .asp, .jsp, .php и др.), если сформирована нужная структура директорий. Конечно же, если для формирования HTML-страниц используется серверное программирование, можно использовать его возможности для того, чтобы вписывать в страницу меню до того, как она покинет сервер.

Далее следует перечень вещей, необходимых для реализации и поддержки предложенного рецепта.

- В каждой странице имеется осмысленный тег <title>.
- В каждой директории должна быть основная страничка (index.html или default.html), которая выдается сервером, если URL содержит директорию.
- Каждый файл должен включать библиотеку trail.js, содержащую данные и процедуры.
- Изменения и добавления в структуре директорий должны отражаться в определении объекта trailMenu библиотеки trail.js.

В этой программе для установления соответствия между именем директории и ее осмысленным названием используется объект, названия свойств которого совпадают с названиями директорий. Обычно в такой ситуации принято использовать массив, но применение объекта позволяет значительно ускорить поиск. Порядок, в котором свойства добавляются к объекту, не важен, но лучше поддерживать его в соответствии со структурой директорий, это упростит поддержку сценария, если в будущем директории будут добавляться, удаляться или переименовываться.

Возможности применения таблиц стилей для управления внешним видом меню практически неограниченны. Кроме того, можно убрать определение стиля из тега и присоединить их к странице в виде отдельного файла .css, снабдив контейнер меню идентификатором или именем класса. В дополнение к этому, можно разместить меню в теле страницы, в таблице или в позиционируемом элементе, как это делается в рецепте.

Чуть больше усилий требуется, чтобы применить этот рецепт при использовании фреймов, когда меню находится в одном фрейме, а просматриваемый пользователем документ — в другом. При этом алгоритм остается прежним, но нужно учитывать два особых случая. Во-первых, необходимо каким-то образом запускать переписывающий меню код каждый раз, когда пользователь переходит на новую страницу. Это можно сделать, указав в каждой странице обработчик события onload. Обработчик должен вызывать функцию (загружаемую из внешней библиотеки .js), которая бы переписывала содержимое меню в другом фрейме (основываясь на значении свойства location). Кроме того, нужно так изменить содержащиеся в меню ссылки, чтобы новые страницы загружались в нужном фрейме.

В показанном рецепте в качестве разделителя между уровнями иерархии используется символ, выглядящий как маленькая двойная стрелка (в действительности это правая угловая кавычка). Можно использовать для этого и другие, более похожие на стрелку символы, например такие, как &arr или &rArr, но они не поддерживаются ни IE для Windows (вплоть до версии 6), ни Netscape

до версии 7 (хотя поддерживаются в IE 5 для Macintosh). Но в качестве разделителя можно использовать и любой другой HTML-код, в том числе даже изображение (для этого нужно просто поместить HTML-код изображения в переменную separator).

Смотрите также

Как формировать длинные строки из фрагментов и кусочков, описывается в рецепте 1.1. Использование document.write() во время загрузки страницы показывается в рецепте 14.1.

10.10. Вложенные меню

NN6 **IE5**

Задача

Нужно создать меню, которое бы выглядело и работало так же, как древовидный список, имеющийся в левой части окна у многих популярных программных продуктов (Windows Explorer, Outlook Express, закладки в Adobe Acrobat и т. д).

Решение

Чтобы поместить в HTML-контейнер вложенное меню, подобное показанному на рис. 10.4, нужно использовать библиотеку expandableMenu.js, код которой приведен в обсуждении в листинге 10.6. Все, что нужно включить в HTML-код страницы, — простой, незаполненный элемент div:

```
<div id="content"></div>
```

В теле страницы нужно назначить обработчиком событии onload функцию инициализации меню initExpMenu():

```
Onload="initExpMenu()";
```

Кроме того, нужно настроить в соответствии с вашими нуждами следующие вещи:

- Изображения элементов оформления меню.
- Значения глобальных переменных, используемых для предварительной загрузки изображений.
- Описание меню в объекте olData.
- Заранее развернутые пункты (необязательно).
- Размеры в таблице стилей, которые нужно привести в соответствие с размерами изображений и шрифтов.
- Этот рецепт работает в Internet Explorer 5 и старше, а также в Netscape 6 и старше. В Opera он в текущем виде не работает, дополнительные сведения по этому вопросу есть в обсуждении.

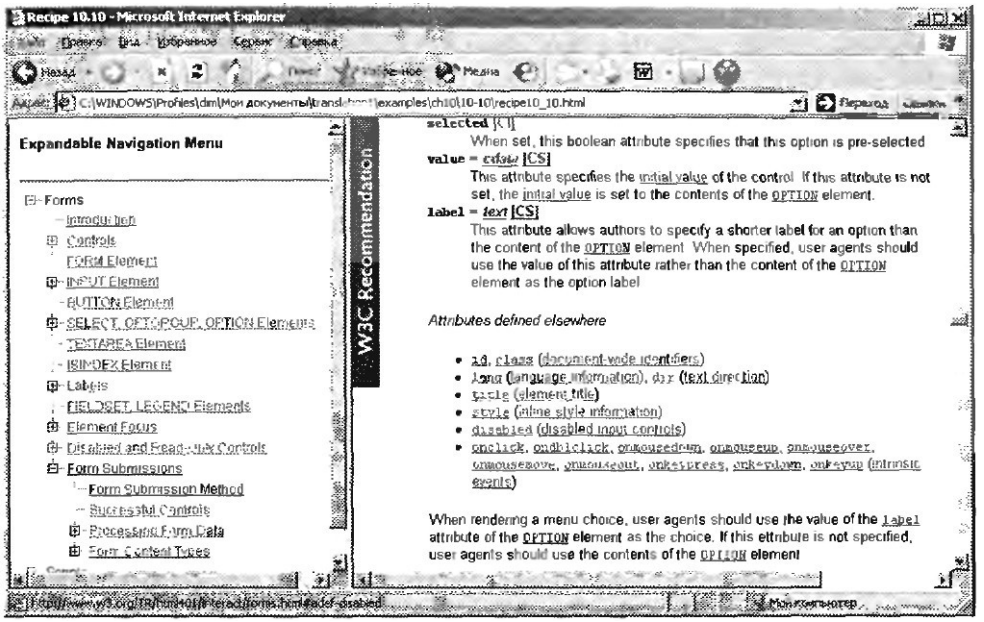


Рис. 10.4. Вложенные меню навигации

Обсуждение

В этом рецепте используется таблица стилей, описывающая внешний вид и расположение создаваемых «на лету» элементов. Нужные правила можно включить в HTML-код страницы или импортировать из файла:

```
<style type="text/css">
    .row {vertical-align:middle; font size:12px; font-family:Arial,
      sans-serif
     }
    .OLBlock {display:none}
    img.widgetArt {vertical-align:text-top}
</style>
```

Библиотека expandableMenu.js приведена в листинге 10.6. Это довольно большая библиотека, разделенная на несколько секций. Показанная здесь версия содержит сокращенный набор данных для меню, показывающего отдельные порции спецификации W3C HTML 4.01.

Листинг 10.6. Библиотека expandableMenu.js

Глобальные переменные

```
// предварительная загрузка изображений и установка размеров элементов
// и разделителей
// (все изображения должны быть одной ширины и высоты)
var collapsedWidget = new Image(20, 16);
```

```
collapsedWidget.src = "oplus.gif";
var collapsedWidgetStart = new Image(20, 16);
collapsedWidgetStart.src = "oplusStart.gif"
var collapsedWidgetEnd = new Image(20, 16);
collapsedWidgetEnd.src = "oplusEnd.gif";
var expandedWidget = new Image(20, 16);
expandedWidget.src = "ominus.gif";
var expandedWidgetStart = new Image(20, 16);
expandedWidgetStart.src = "ominusStart.gif"
var expandedWidgetEnd = new Image(20, 16);
expandedWidgetEnd.src = "ominusEnd.gif";
var nodeWidget = new Image(20, 16);
nodeWidget.src = "onode.gif";
var nodeWidgetEnd = new Image(20, 16);
nodeWidgetEnd.src = "onodeEnd.gif";
var emptySpace = new Image(20, 16);
emptySpace.src = "oempty.gif";
var chainSpace = new Image(20, 16);
chainSpace.src = "ochain.gif";

// разные глобальные переменные
var widgetWidth = "20";
var widgetHeight = "16";
var currState = "";
var displayTarget = "contentFrame";


          Данные меню

var expansionState = "";
// конструктор объектов для контурных линий
function outlineItem(text, uri) {
    this.text = text;
    this.uri = uri;
}
var olData = {childNodes:
             [{item:new outlineItem("Forms"),
               childNodes:
                   [{item:new outlineItem("Introduction",
                  ".../forms.html#h-17.1")},

                    {item: new outlineItem("INPUT Element",
                   ".../forms.html#h-17.4"),
                     childNodes:
                         [{item:new outlineItem("INPUT Control Types",
                        " ../forms.html#h-17.4.1")},
                          {item:new outlineItem("Examples",
                         ".../forms.html#h-17.4.2")}
                          ]},


             },
              {item:new outlineItem("Scripts"),
               childNodes:
                   [{item: new outlineItem("Introduction",
                  ".. ./scripts.html#h-18.1")},
```

продолжение ⇙

Листинг 10.6 (продолжение)

```
                {item:new outlineItem("Designing Documents for Scripts"
              ".../scripts html#h-18.2"),
                childNodes:
                    [{item:new outlineItem("SCRIPT Element".
                  ".../scripts.html#h-18.2.1")},
                      {item:new outlineItem("Scripting Language".
                    ".../scripts.html#h-18.2.2"),
                      childNodes:
                      [{item:new outlineItem("Default Language".
                  ".../scripts.html#h-IB.2.2.1")}.
                      {item:new outlineItem("Local Language
    Declaration". ".../scripts.html#h-IB.2.2.2")}.
                          {item:new outlineItem("References to HTML
    Elements". ".../scripts html#h-18.2.2.3")}
                          ]}.
                  ]

            Г
            ]}
          ]
      }:


  Управление видимостью ветвей и видом значков
A*********************************************/
// инвертирование состояния элемента (развернутое/свернутое)
function swapState(currState. currVal. n) {
    var newState - currState.substring(0.n);
    newState += currVal  ^ 1 //bitwise XOR item n:
    newState +- currState.substring(n+1.currState.length);
    return newState:
}

// определяем подходящее изображение со значком 'минус'
function getExpandedWidgetState(imgURL) {
    if (imgURL.indexOf("Start") !- -1) {
        return expandedWidgetStart.src;
    }
    if (imgURL.indexOf("End") !- -1) {
        return expandedWidgetEnd.src;
    }
    return expandedWidget.src;
}

// определяем подходящее изображение со значком 'плюс'
function getCollapsedWidgetState(imgURL) {
    if (imgURL.indexOf("Start") != -1) {
        return collapsedWidgetStart.src;
    }
    if (imgURL.indexOfC'End") !.= -1) {
        return collapsedWidgetEnd.src;
    }
    return collapsedWidget.src;
```

```
// Изменение состояния родительского элемента меню и сохранение нового
// состояния. Вызывается обработчиками события click из элементов
// управления
function toggle(img. blockNum) {
    var newString = "":
    var expanded. n:
    // изменяем строку состояния. основываясь на параметрах из IMG
    expanded = currState.charAt(blockNum);
    currState = swapState(currState. expanded. blockNum):
    // динамически изменяем стиль отображения
    if (expanded == "O") {
        document.getElementById("OLBlock" + blockNum).style.display="block";
        img.src = getExpandedWidgetState(img.src);
    } else {
        document.getElementById("OLBlock" + blockNum).style.display= "none":
        img.src = getCollapsedWidgetState(img.src);
    }
}

function expandAll() {
    var newState = "":
    while (newState.length< currState.length) {
        newState += "1":
    }
    currState - newState:
    initExpand0:
}

function collapseAll() {
    var newState - "":
    while (newState.length< currState.length) {
        newState +- "0":
    }
    currState - newState;
    initExpand0;
}

/  Л  Л-А  *  ****************************
   Генерация HTML-кода меню

// Применяем разворачивание по умолчанию для одного из элементов меню,
// чтобы облегчить инициализацию переменной currState
function calcBlockState(n) {
    // Определяем состояние по умолчанию
    var expandedData - (expansionState.length > 0) ?
       expansionState.split(",") • null:
    if (expandedData) {
        for (var j - 0: j < expandedData.length; j++) {
            if (n == expandedData[j] - 1) {
                return "1":
            }
        }
    }
    return "0";
```

продолжение

Листинг 10.6 (продолжение)

```
// счетчики обратных вызовов drawOutline()
var currID = 0;
var blockID = 0;
// Генерация HTML-кода меню
function drawOutline(ol, prefix) {
    var output = "";
    var nestCount, link, nestPrefix, lastInnerNode;
    prefix = (prefix) ? prefix : ''
    for (var i = 0; i < ol.childNodes.length ; i++) {
        nestCount = (ol.childNodes[i].childNodes) ?
          ol.childNodes[i].childNodes.length : 0;
        output += "<div class='OLRow' id='line" + currID++ + "'>\n";
        if (nestCount > 0) {
            output += prefix;
            output += "<img id='widget" + (currID-1) + "' src='" +
              ((i == ol.childNodes.length-1 && blockID != 0) ?
                 collapsedWidgetEnd.src :
              (blockID == 0) ? collapsedWidgetStart.src :
                 collapsedWidget.src);
            output += "' height=" + widgetHeight +  " width=" + widgetWidth;
            output += " title='Click to expand/collapse nested items.'"+
                   "onClick='toggle(this," + blockID + ")'> ";
            link =  (ol.childNodes[i].item.uri) ? ol.childNodes[i].item.uri:

            if (link) {
              output += "<a href='" + link + "' class='itemTitle' title='" +
                link + "' target='" + displayTarget + "'>"
            } else {
                output += "<a class='itemTitle' title='" + link + "'>";
            }
            output+="<span style='position:relative;top:-3px;height:11px'>"
              + ol.childNodes[i].item.text + "</span></a>";
            currState += calcBlockState(currID-1);
            output += "<span class='OLBlock' blocknum='" + blockID +"'"+
              "id='OLBlock" + blockID++ + "'>"
            nestPrefix = prefix;
            nestPrefix += (i == ol.childNodes.length - 1) ?
                       "<img src='" + emptySpace.src + "' height=" +
                         widgetHeight + " width=" + widgetWidth + ">"
                       "<img src='" + chainSpace.src + "' height=" +
                         widgetHeight + " width=" + widgetWidth + ">"
            output += drawOutline(ol.childNodes[i], nestPrefix);
            output += "</span></div>\n";
        } else {
            output += prefix;
            output += "<img id='widget" + (currID-1) + "' src='" +
              ((i == ol.childNodes.length - 1) ? nodeWidgetEnd.src :
                 nodeWidget.src);
            output+="' height="+widgetHeight+" width=" + widgetWidth + ">";
            link=(ol.childNodes[i].item.uri) ? ol.childNodes[i].item.uri:"";
            if (link) {
                output+=" <a href='"+link+"' class='itemTitle' title='"
              +link + "' target='" + displayTarget + "'>";
            } else {
```

```
                output+= " <a class='itemTitle' title='" + link + "'>";

            output += "<span style='position:relative;top:-3px;"
             "height:11px'>" + ol.childNodes[i].item.text + "</spanx/a>".
            output += "</div>\n";

        )
    }
    return output;
}

y*********************************
    Инициализация меню

// разворачиваем пункты, установленные в переменной expansionState. если
// такие есть
function initExpand0 {
    for (var i = 0; i < currState.length; i++) {
        if (currState.charAt(i) == 1) {
            document.getElementById("OLBlock" + i).style.display = "block";
        } else {
            document.getElementById("OLBlock" + i).style.display = "none";
        }
    }


// первичная инициализация, вызывается из onload
function initExpMenu(xFile) {
    // заворачиваем весь HTML-код меню в span
    var olHTML = "<span id='renderedOL'>" + drawOutline(olData) + "</span>";
    // помещаем HTML-код в div 'content', чтобы отобразить его
    document.getElementById("content").innerHTML = olHTML;
    initExpand0;
}
```

Сценарий в этом примере начинается с объявления и предварительной загрузки маленьких изображений, которые станут элементами оформления. Изображения, использованные в этом рецепте, показаны на рис. 10.5. Все они должны иметь одинаковый размер.

Каждый объект Image помещается в глобальную переменную, как и другие значения, часто используемые при генерации HTML и взаимодействии с пользователем. К примеру, глобальная переменная **currState** хранит состояние различных элементов меню (свернутый/развернутый).

Следующий блок кода содержит данные о меню и некоторые вспомогательные значения. Одна из глобальных переменных, expansionState, содержит номера строк элементов, которые должны быть развернуты при первом показе меню (каждый элемент находится на своей строке). Если сначала должны быть видимы только элементы верхнего уровня (то есть меню полностью свернуто), в эту переменную нужно поместить пустую строку. При загрузке страницы многократно вызывается конструктор **outlineItem()**, формирующий объекты меню, каждый из которых имеет название, и необязательно соответствующий URL.

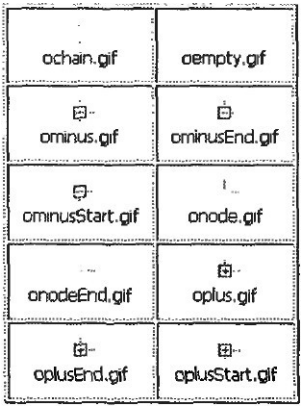


Рис. 10.5. Изображения, используемые для рисования меню

Далее следует определение объекта, содержащего информацию о структуре меню (olData). Для этого примера выбрана структура, в своей основе имитирующая узлы XML. Итак, последовательность вложенных объектов и массивов определяет структуру меню. В этом примере приведена только часть меню (ссылки на отдельные секции рекомендаций W3C HTML 4.01). Более подробно оформление будет рассмотрено позднее.

Секция, названная «Управление видимостью ветвей и видом значков», включает в себя функции, управляющие сворачиванием и разворачиванием пунктов. Две функции, getExpandedWidgetState() и getCollapsedWidgetState() (обе они вызываются из функции toggle(), которая рассматривается позже), служат для получения одного из трех развернутых или свернутых изображений. Исходными данными для них служит имя (точнее, часть имени) изображения текущего элемента. Вспомогательная функция swapState() (которая также вызывается из toggle()) выполняет двоичные вычисления со значением переменой currState, изменяя значение отдельных символов с нуля на единицу, и наоборот (эти символы означают состояние отдельных узлов).

В основе взаимодействия с пользователем лежит функция toggle(), вызываемая из обработчиков события onclick у всех реагирующих на нажатия элементов оформления. Так как обработчики событий назначаются при формировании HTML-кода меню, они могут включать параметры, позволяющие определить, какой пункт был выбран пользователем. Так, обработчик события получает URL изображения текущего элемента оформления (по этому параметру определяется, какое изображение должно его заменить) и числовой идентификатор элемента span, содержащего вложенные пункты. Эта функция достаточно мала, но использует в работе вспомогательные процедуры. Две ее основные задачи — это замена изображения элемента оформления и изменение атрибута стиля display у содержащего вложенные пункты элемента.

Еще две функции, expandAll() и collapseAll(), предназначены для сворачивания или разворачивания всех пунктов, если ваш пользовательский интерфейс предполагает такую возможность.

Предпоследний блок посвящен формированию HTML-кода меню. Вспомогательная функция calcBlockState(), многократно вызываемая при конструировании

HTML, и проверяет, не должна ли строка с определенным номером быть развернута по умолчанию. Эти данные хранятся в виде разделенного запятыми списка номеров развернутых строк (хранящегося в глобальной переменной expansionState).

В процессе сборки HTML функция drawOutline() перебирает элементы массива объекта olData. При этом производятся рекурсивные вызовы той же самой функции drawOutline(), формирующие код вложенных элементов. Чтобы назначать элементам уникальные идентификаторы, используются две глобальные переменные, играющие роль счетчиков: currID и blockID.

Итак, теперь рассмотрим функцию drawOutline(), которая, действуя подобно кружащемуся дервишу, накапливает HTML-код меню. Код меню формируется только один раз, а все последующие сворачивания и разворачивания элементов производятся путем манипуляций таблицами стилей. Расположением отдельных элементов оформления управляет структура объекта olData. Помимо других, более сложных задач, функция должна следить, нужно ли вставлять перед изображением вертикальную линию, означающую предыдущие соединения, или же оставлять пространство пустым. Одинаковость размеров всех изображений позволяет сценарию строить каждую линию из отдельных изображений, как если бы они были плитками мозаики.

Заключительный блок кода выполняет всю инициализацию и приводит систему в движение. Первая функция (initExpand()) перебирает все элементы переменной currState и устанавливает требуемое состояние (свернутый/развернутый) для всех узлов. Эта функция вызывается не только из initExpMenu(), но и из функций expandAll() и collapseAll().

Наконец, рассмотрим функцию initExpMenu(), вызываемую страницей из обработчика события onload. Эта функция является движущей силой, приводящей в действие формирование HTML-кода, внедрение этого кода в выделенный элемент span, и доставляет все это в предварительно объявленный в коде страницы элемент div.

Для успешного применения этой системы меню нужно создать свой собственный набор изображений для элементов управления, подобных показанным на рис. 10.5. Все изображения должны иметь одинаковый размер, а если вы используете размеры, отличные от показанных в рецепте, в таблице стилей нужно отрегулировать размеры шрифтов, чтобы добиться пропорционального вида.

Большую часть изменений нужно вносить в верхнюю часть кода сценария. Начать изменения можно с указания URL для изображений, размеров изображений и фреймов (предложенная система меню лучше всего работает при использовании фреймов или элементов iframe, чтобы меню не перезагружалось при переходах).

Пожалуй, наиболее сложная часть настройки — создание объекта olData. Прежде чем приступать к этой части настройки, нужно подготовить иерархию меню. Выписать ее не так уж сложно, зато это поможет визуализировать вложение. Например, начало показанного в примере меню будет полностью в развернутом виде:

```
Forms
    Introduction
    Controls
       Control Types
    FORM Element
```

```
INPUT Element
    INPUT Types
    Examples
BUTTON Element
```

Каждый пункт меню находится на отдельной строке. Каждому пункту должно быть сопоставлено его название и URL. Если URL не указывать, пункт все равно появится в меню, но с ним не будет связано никакого документа.

Теперь, чтобы преобразовать выписанный текст в определение объекта olData, нужно установить иерархические отношения между пунктами. Схематически показанный выше фрагмент иерархии может быть переделан в определение объекта следующим образом:

```
olData =
  {childNodes:[{item:"Forms",
                childNodes:[{item:"Introduction"},
                            {item:"Controls",
                             childNodes:[{item:"Control types"}]}
                            {item:"FORM Element"},
                            {item:"INPUT Element",
                             childNodes:[{item:"INPUT types"},
                                         {item:"Examples"}]}
                            {item:"BUTTON Element"}]
               }]
  };
```

Самая сложная часть работы — поддержание правильного вложения объектов (то есть правильного расположения открывающих и закрывающих скобок) по мере роста меню. Свойство **childNodes**, которое используется в этом рецепте, свое начало берет из XML-версии этого меню, показанной в рецепте 10.11. Но вы можете использовать любое другое имя для этого свойства. Кроме того, оформление кода отступами упрощает поддержание порядка вложения.

Номера строк в полностью развернутом меню могут быть важны, если нужно, чтобы некоторые пункты были бы развернуты по умолчанию. Разделенный запятыми список номеров развернутых строк нужно поместить в глобальную переменную expansionState. Туда нужно помещать номера только тех строк, которые служат узлами ветвления, то есть содержат вложенные пункты. Если же поместить в переменную пустую строку (как это и сделано в рецепте), то изначально будут видны только пункты верхнего уровня.

Несколько иначе работает переменная **currState**. Ее значение представляет собой строку из единиц и нулей, отражающих состояние только узлов ветвления. То есть если меню содержит пять узлов ветвления и несколько дюжин конечных узлов (то есть пунктов, не имеющих дочерних узлов), то значение в переменной состоит из пяти цифр. Нулевое значение в одном из разрядов означает, что соответствующий узел свернут, единица — что узел развернут. Такой механизм позволяет легко изменять состояние узла на обратное в ответ на щелчок мышью (это делается в функции **swapState()**). Это также позволяет запоминать, какие из ветвей более низкого уровня развернуты. Другими словами, если узел, занимающий более высокий уровень, будет свернут, то после его разворачивания вложенные узлы сохранят свое состояние. Кроме того, если вы хотите

сохранять состояние узлов между посещениями сайта, вы должны сохранять значение переменной currState в cookie и восстанавливать его при посещениях. Для пользователей, впервые посетивших сайт, нужно предоставить значение по умолчанию.

В приведенном рецепте видимыми узлами дерева ссылок является только текст, но это не является ограничением. Объект olData можно изменить так, чтобы в нем хранился любой HTML-код, помещаемый на место пунктов меню. Циклом drawOutline() все так же будет управлять перечисление элементов объекта olData.

Может возникнуть вопрос: зачем при первом вызове drawOutline() из initExpMenu() в нее передается ссылка на объект olData? Дело в том, что функция drawOutline() рекурсивна по своей природе, то есть повторно вызывает сама себя, передавая в качестве аргументов вложенные фрагменты olData. Поэтому удобно считать, что функция всегда получает параметр, содержащий действительную ссылку на элемент olData. При этом при первом вызове аргумент ссылается на весь объект olData целиком, а при последующем спуске к более глубоко вложенным элементам аргумент ссылается на все меньшие его части. Никаких других признаков первого запуска или дополнительных проверок завершения цикла передавать не нужно.

Если вы собираетесь использовать этот рецепт, не генерируя дерево ссылок динамически, а жестко закодировав его в HTML-коде (например, чтобы поисковые машины могли видеть ссылки и следовать им), вы все равно можете применять объект olData для организации данных дерева ссылок. Для этого нужно загрузить страницу в браузер и зафиксировать HTML-код дерева ссылок. Самый простой способ добиться этого — использовать программированные закладки (bookmarklet), то есть закладки, содержащие URL вида javascript:, выполняющий некоторый код. Или же можно просто ввести в строку адреса браузера такой URL:

```
javascript: "<textarea cols='120' rows='40'>" +
document.getElementById("content").innerHTML + "</textarea>"
```

После этого появится элемент textarea, содержащий целиком HTML-код дерева ссылок. Этот код можно скопировать и вставить в элемент div, предназначенный для размещения меню (с идентификатором content). (Такая методика может быть полезна и для получения с целью изучения или отладки кода, генерируемого функцией drawOutline().) Затем в глобальную переменную currState нужно поместить строку из нулей, а блоки кода, названные «Данные меню» и «Генерация HTML-кода меню», можно удалить, кроме вызова initExpand() из функции initExpandMenu(). Если меню содержит тысячи пунктов (а это говорит о том, что меню чересчур велико), то, напрямую закодированное в HTML, оно отобразится быстрее, чем динамически генерируемая версия.

Смотрите также

Построение больших строк из маленьких фрагментов смотрите в рецепте 1.1. Древовидное меню, основанное на XML, показано в рецепте 10.11. Как предварительно загрузить изображения, показано в рецепте 12.1. Управление видимостью элементов демонстрируется в рецепте 12.7.

10.11. Сворачиваемое меню на основе XML

NN6 **IE5(Win)**

Задача

Нужно создать меню, которое бы выглядело и работало так же, как древовидный список, имеющийся в левой части окна у многих популярных программных продуктов (Windows Explorer, Outlook Express, закладки в Adobe Acrobat и т. д.), а данные для меню должны храниться в XML.

Решение

Библиотека XMLoutline.js, показанная в обсуждении, поможет сформировать интерактивное древовидное меню, данные для которого берутся из специально оформленного документа XML. Пример такого меню приведен на рис. 10.4 в рецепте 10.10. Там, где в документе будет располагаться будущее меню, нужно добавить пустой элемент div:

```
<div id="content"></div>
```

В теле страницы нужно вызвать функцию initXMLOutline() из обработчика события onload страницы. Функции нужно передать адрес XML-файла:

```
Onload="initXMLOutline('SpecOutline.xml')";
```

Кроме того, в нижней части страницы нужно добавить тег <object>, загружающий нужный элемент управления ActiveX в Internet Explorer для Windows перед его использованием в коде сценария:

```
<!-Попытка загрузить объект Msxml.DOMDocument ActiveX для поддержки
   верификации ->
<object id="msxml" WIDTH="1" HEIGHT="1"
   classid="CLSID:2933BF90-7B36-11d2-B20E-00C04F983E60" ></object>
```

Нужно также изменить в соответствии с вашими требованиями следующие вещи:

- источник данных OPML;
- изображения для оформления меню;
- глобальные переменные в сценарии для предварительной загрузки изображений;
- размеры в таблицах стилей нужно привести в соответствие с размерами изображений.

Как это сделать, подробнее описывается в обсуждении. Этот рецепт работает в Internet Explorer 5 для Windows, Netscape 6 и старше.

Обсуждение

Оказанный здесь рецепт во многом подобен основанному на JavaScript решению из рецепта 10.10. Разница в том, что структура меню описана на XML, приспо-

собленном для описания деревьев: OPML (Outline Processing Markup Language), разработанном Dave Wiener (http://www.opml.org). Поэтому код, ответственный за сворачивание и разворачивание, идентичен коду в рецепте 10.10, но создание и загрузка меню выглядит по-другому. Несмотря на общие детали, этот рецепт для полноты будет рассмотрен отдельно.

В этом рецепте используется таблица стилей, описывающая внешний вид и расположение создаваемых «на лету» элементов. Нужные правила можно включить в HTML-код страницы или импортировать из файла:

```
<style  type="text/css">
    .row {vertical-align:middle: font-size: 12px: font - family: Aria l.
       sans-serif}
    .OLBlock {display:none}
    img.widgetArt {vertical-align:text-top}
</style>
```

Библиотека **XMLOutline.js** приведена в листинге 10.6. Так как описание меню хранится в отдельном файле, библиотека целиком состоит из интерактивного кода.

Листинг 10.7. Библиотека XMLOutline.js

```
            Глобальные переменные

// Предварительная загрузка изображений для оформления меню
// (все изображения должны быть одинаковой высоты и ширины)
var collapsedWidget = new Image(20. 16):
collapsedWidget.src = "oplus.gif";
var collapsedWidgetStart = new Image(20. 16):
collapsedWidgetStart.src = "oplusStart.gif":
var collapsedWidgetEnd = new Image(20. 16):
collapsedWidgetEnd.src = "oplusEnd.gif":
var expandedWidget = new Image(20, 16):
expandedWidget.src = "ominus.gif":
var expandedWidgetStart = new Image(20. 16):
expandedWidgetStart.src = "ominusStart.gif":
var expandedWidgetEnd = new Image(20. 16);
expandedWidgetEnd.src = "ominusEnd.gif";
var nodeWidget - new Image(20, 16):
nodeWidget.src - "onode.gif";
var nodeWidgetEnd - new Image(20. 16):
nodeWidgetEnd.src - "onodeEnd.gif":
var emptySpace = new Image(20, 16):
emptySpace.src - "oempty.gif";
var chainSpace - new Image(20, 16):
chainSpace.src - "ochain.gif":

// разные глобальные переменные
var widgetWidth - "20":
var widgetHeight - "16":
var currState - "";
var displayTarget - "contentFrame";
// объект XML-документа
var xDoc:
```

продолжение

Листинг 10.7 (продолжение)

Управление отображением и значками

```
// инвертируем состояние элемента (свернутый/развернутый)
function swapState(currState. currVal, n) {
    var newState = currState.substring(O.n):
    newState += currVal  ^ 1 //Bitwise XOR item n:
    newState += currState.substring(n+l.currState.length);
    return newState:
}

// определяем соответствующую версию значка "минус"
function getExpandedWidgetState(imgURL) {
    if (imgURL.indexOf("Start") != -1) {
        return expandedWidgetStart.src:
    }
    if (imgURL.indexOf("End") != -1) {
        return expandedWidgetEnd.src:
    }
    return expandedWidget.src:
}

// определяем соответствующую версию значка "плюс"
function getCollapsedWidgetState(imgURL) {
    if (imgURL.indexOf("Start") != -1) {
        return collapsedWidgetStart.src:
    }
    if (imgURL.indexOf("End") != -1) {
        return collapsedWidgetEnd.src:
    }
    return collapsedWidget.src:
}

// по событию onclick в элементе оформления меню инвертируем состояние
// родительского пункта меню и сохраняем новое значение
function toggle(img, blockNum) {
    var newString = "";
    var expanded, n:
    // Основываясь на параметрах IMG. меняем строку состояния.
    expanded = currState.charAt(blockNum);
    currState - swapState(currState. expanded. blockNum):
    // динамически меняем стиль отображения
    if (expanded — "0") {
        document.getElementById("OLBlock" + blockNum).style.display="block"
        img.src = getExpandedWidgetState(img.src);
    } else {
        document.getElementById("OLBlock" + Ы ockNum).style.di splay="none":
        img.src = getCollapsedWidgetState(img.src):
    }
}

function expandAll() {
    var newState - "";
    while (newState.length< currState.length) {
```

```
        newState += "1";
    currState = newState;
    initExpand();
}
function collapseAll() {
    var newState = "";
    while (newState.length < currState.length) {
        newState += "0";
    }
    currState = newState;
    initExpand();
}
/*********************************
    Генерация HTML-кода меню
// Применяем разворачивание по умолчанию для одного из элементов меню.
// чтобы упростить инициализацию переменной currState
function calcBlockState(n) {
    var ol = xDoc.getElementsByTagName("body")[0];
    var outlineLen = ol.getElementsByTagName("outline").length;
    // Считываем значение OPML expansionState
    var expandElem = xDoc.getElementsByTagName("expansionState")[0];
    var expandedData = (expandElem.childNodes.length) ?
       expandElem.firstChild.nodeValue.split(",") : null;
    if (expandedData) {
        for (var j = 0; j < expandedData.length; j++) {
            if (n == expandedData[j] - 1) {
                return "1";



    return "0";
}

// счетчики обратных вызовов drawOutline()
var currID = 0;
var blockID = 0;
// генерация HTML-кода меню
function drawOutline(ol, prefix) {
    var output = "";
    var nestCount, link, nestPrefix, lastInnerNode;
    ol = (ol) ? ol : xDoc.getElementsByTagName("body")[0];
    prefix = (prefix) ? prefix : "";
    if (ol.childNodes[ol.childNodes.length - 1].nodeType == 3) {
        ol.removeChild(ol.childNodes[ol.childNodes.length - 1]);

    for (var i = 0; i < ol.childNodes.length; i++) {
        if (ol.childNodes[i].nodeType == 3) {
            continue;
        }
        if (ol.childNodes[i].childNodes.length > 0 &&
            ol.childNodes[i].childNodes[ol.childNodes[i].childNodes.length-1]
               .nodeType == 3) {
```

продолжение

Листинг 10.7 (продолжение)

```
            ol.childNodes[i].removeChild(ol.childNodes[i].childNodes[ol.childNodes[i]
               .childNodes.length - 1]);
        }
        nestCount - ol.childNodes[i].childNodes.length;
        output +- "<div class='OLRow' id='line" + currID++ + "'>\n";
        if (nestCount > 0) {
            output += prefix;
            output += "<img id='widget" + (currID-1) + "' src='" + ((i==
               ol.childNodes.length-1) ? collapsedWidgetEnd.src : (blockID==0) ?
                  collapsedWidgetStart.src : collapsedWidget.src);
            output += "' height=" + widgetHeight + " width=" + widgetWidth;
            output += " title='Click to expand/collapse nested items.'
               onClick='toggle(this," + blockID + ")'>";
            link =  (ol.childNodes[i].getAttribute("uri")) ?
               ol.childNodes[i].getAttribute("uri") : "";
            if (link) {
                output += " <a href-'" + link + "' class='itemTitle' title='" +
                link + "' target='" + displayTarget + "'>" :
            } else {
                output+= " <a class='itemTitle' title-'" + link + "'>";
            }
            output += "<span style='position:relative; top:-3px; height:11px'> "
               + ol.childNodes[i].getAttribute("text") + "</span></a>";
            currState += calcBlockState(currID-1);
            output += "<span class='OLBlock' blocknum='" + blockID + "' id='OLBlock"
               + blockID++ + "'>";
            nestPrefix - prefix;
            nestPrefix += (i == ol.childNodes.length - 1) ?
                      "<img src='" + emptySpace.src + "' height=" +
                    widgetHeight + " width=" + widgetWidth + ">"
                      "<img src='" + chainSpace.src + "' height=" +
                    widgetHeight + " width=" + widgetWidth + ">"
            output += drawOutline(ol.childNodes[i], nestPrefix);
            output += "</span></div>\n";
        } else {
            output += prefix;
            output += "<img id='widget" + (currID-1) + "' src='" + ((i --
               ol.childNodes.length - 1) ? nodeWidgetEnd.src : nodeWidget.src);
            output+="' height="+ widgetHeight+ " width="+ widgetWidth + ">";
            link =  (ol.childNodes[i].getAttribute("uri")) ?
              ol.childNodes[i].getAttribute("uri") : "";
            if (link) {
                output+=" <a href='"+link+"' class='itemTitle' title='"
                     + link + "' target='" + displayTarget + '">";
            } else {
                output +=" <a class='itemTitle' title-'" + link + '">";
            }
            output +="<span style='position:relative; top:-3px; height:11px'>"
               +  ol.childNodes[i].getAttribute("text") + "</span></a>";
            output += "</div>\n";
        }
    }
    return output;
}
```

Инициализация меню

```
// разворачиваем пункты, перечисленные в OPML-теге expansionState. если
// таковые есть
function initExpand() {
    for (var i - 0; i < currState.length; i++) {
        if (currState.charAt(i) -- 1) {
            document.getE ementById("OLBlock" + i).style.display = "block":
        } else {
            document getElementById("OLBlock" + i).style.display = "none":
        }
    }
}
function finishInit() {
       // извлекаем элемент описания меню body для дальнейшей обработки
       // и преобразования в HTML
        var ol = xDoc.getElementsByTagName("body")[0]:
       // Заворачиваем весть HTML-код меню в span
        var olHTML = "<span id='renderedOL'>" + drawOutline(ol) + "</span>":
       // перебрасываем HTML-код в div 'content', чтобы отобразить его
        document.getElementById("content").innerHTML = olHTML:
        initExpandO:
}

function continueLoad(xFile) {
    xDoc.load(escape(xFile));
    // IE перед чтением требуется задержка, чтобы содержимое успело
    // загрузиться
    setTimeout("finishInit()" 300):
}

// проверка, поддерживает ли браузер XML и загрузил ли внешний .xml файл
function loadXMLDoc(xFile) {
    if (document.implementation&& document.implementation.createDocument) {
       // метод W3C DOM, поддерживаемый пока что только в NN6
        xDoc = document.implementation.createDocument("", "theXdoc", null):
    } else if (typeof ActiveXObject != "undefined") {
       // проверяем, поддерживается ли настоящий объект (прости, IE5/Mac)
        if (document.getElementById("msxml").async) {
            xDoc = new ActiveXObject("Msxml.DOMDocument");
        }
    }
    if (xDoc && typeof xDoc.load != "undefined") {
       // Netscape 6+ для загрузки требуется эта задержка.
       // начинаем последовательность из двух этапов
        setTimeout("continueLoad('" xFile + " ' ) " . 50):
    } else {
        var reply = confirm("Этот пример требует браузера, поддерживающего XML, такого как
IE5+/Windows или Netscape 6+\n \тВернуться к предыдущей странице?"):
        if (reply) {
            history.backO:
        }
```

продолжение

Листинг 10.7 (продолжение)

```
}

// начальная инициализация, вызывается из onload
function initXMLOutline(xFile) {
    loadXMLDoc(xFile):
}
```

Сценарий начинается с объявления и предварительной загрузки маленьких изображений, которые станут элементами оформления древовидного меню. Используемые в этом рецепте изображения показаны на рис. 10.5 в рецепте 10.10. Все изображения должны иметь одинаковые размеры. Каждому изображению соответствует глобальная переменная, как и некоторым другим объектам, включая переменную xDoc, содержащую ссылку на скрытый документ XML.

Секция, названная «Управление отображением и значками», содержит несколько функций, предназначенных для изменения состояния ветвей, от свернутого до развернутого. Пара функций getExpandedWidgetState() и getCollapsedWidgetState() (обе они вызываются из функции toggle()) возвращают, в зависимости от имени текущего элемента оформления, одно из трех свернутых или развернутых изображений. Вспомогательная функция swapState() (также вызываемая из toggle()) производит двоичные вычисления, чтобы изменить значение отдельного символа с нуля на единицу или обратно (эти символы означают состояние каждого узла ветвления).

В основе взаимодействии с пользователем лежит функция toggle(), вызываемая из обработчиков события onclick у всех реагирующих на нажатия элементов оформления. Так как обработчики событий назначаются при формировании HTML-кода меню, они могут включать параметры, позволяющие определить, какой пункт был выбран пользователем. Так, обработчик события получает URL изображения текущего элемента оформления (по этому параметру определяется, какое изображение должно его заменить) и числовой идентификатор элемента span, содержащего вложенные пункты. Эта функция достаточно мала, но использует в работе вспомогательные процедуры. Две ее основные задачи — это замена изображения элемента оформления и изменение атрибута стиля display у содержащего вложенные пункты элемента.

Еще две функции, expandAll() и collapseAll(), предназначены для сворачивания или разворачивания всех пунктов, если ваш пользовательский интерфейс предполагает такую возможность.

Предпоследний блок посвящен формированию HTML-кода меню. Вспомогательная функция calcBlockState(), многократно вызываемая при конструировании HTML, проверяет, не должна ли строка с определенным номером быть развернута по умолчанию. Эти данные хранятся в виде разделенного запятыми списка номеров развернутых строк (OPML-атрибут expansionState).

Формирующая HTML-код меню функция drawOutline() перебирает дерево узлов объекта xDoc (описан далее). Но самая большая часть вычислений производится при рекурсивных вызовах drawOutline(), формирующих код для вложенных пунктов. Чтобы генерировать уникальные идентификаторы элементов, используются две глобальные переменные, currID и blockID.

Теперь рассмотрим функцию drawOutline(), формирующую код меню Этот код формируется только один раз, и все последующие сворачивания и разворачивания производятся с помощью таблиц стилей. Расположение элементов HTML-документа задается иерархией элемента xDoc. Помимо других, более сложных задач, функция должна следить, нужно ли вставлять перед изображением вертикальную линию, означающую предыдущие соединения, или же оставлять пространство пустым. Одинаковость размеров всех изображений позволяет сценарию строить каждую линию из отдельных изображений, как если бы они были плитками мозаики.

Заключительный блок кода выполняет всю инициализацию и приводит систему в движение. Первая функция (initExpand()) перебирает все элементы переменной currState и устанавливает требуемое состояние (свернутый/развернутый) для всех узлов. Эта функция вызывается не только из initExpMenu(), но и из функций expandAII() и collapseAII().

Последовательность вызовов трех функций (связанных через setTimeout()) загружает внешний документ XML и запускает функцию drawOutline(). Паузы между вызовами необходимы для того, чтобы обойти некоторые особенности IE и NN, связанные с задержками. Последовательно вызываются три функции: loadXMLDoc(), continueLoad() и finishInit(). В коде сценария эти функции перечислены в порядке, обратном порядку их срабатывания. Выполнение начинается с проверки сценарием того, что браузер поддерживает чтение внешних документов XML (с помощью методик IE/Windows или NN/Netscape). Если проверка прошла успешно, документ загружается в переменную xDoc. На последнем этапе из документа OPML извлекается секция body и передается в функцию генерации меню drawOutline() (хотя к ней можно обращаться и в других местах сценария).

OPML представляет собой расширяемый формат описания меню. Документ OPML разделен на два блока, head и body. Иерархия элементов (их вложение) полностью определяется вложением элементов outline. К любому элементу можно добавлять любые новые атрибуты, и все равно соответствовать формату (если запись атрибутов и значений синтаксически соответствует правилам XML). Далее приведена выжимка из XML-документа, описывающего структуру меню, подобного показанному на рис. 10.4 (URL урезаны с целью экономии места):

```
<?xml version="1.0"?>
<opml version="1.0">
   <head>
      <title>HTML Sections Outline</title>
      <dateCreated>Mon. 10 Sep 2002 03:40:00 GMT</dateCreated>
      <dateModified>Fri. 22 Sep 2002 19:35:00 GMT</dateModified>
      <ownerName>Danny Goodman</ownerName>
      <ownerEmail>dannyg@dannyg.com</ownerEmail>
      <expansionState></expansionState>
      <vertScrollState>1</vertScrollState>
      <windowTop></windowTop>
      <windowLeft></windowLeft>
      <windowBottom></windowBottom>
      <windowRight></windowRight>
   </head>
   <body>
      <outline text="Forms">
```

```
    <outline text="Introduction" uri=".../forms.html#h-17.1"/>
      <outline text="Controls" uri=".../forms.html#h-17.2">
        <outline text="Control Types"
              uri=" ../forms.html#h-17.2.1"/>
      </outline>
      <outline text="FORM Element" uri=".../forms.html#h-17.3"/>
      <outline text="INPUT Element" uri=".../forms.html#h-17.4">
        <outline text="INPUT Control Types"
              uri=" . /forms.html#h-17.4.1"/>
        <outline text="Examples" uri=".../forms.html#h-17.4.2"/>

    </outline>
    <outline text="Scripts">
      <outline text="Introduction" uri=".../scripts.html#h-1B.1"/>
      <outline text="Designing Documents for Scripts"
          uri=".../scripts.html#h-18.2">
        <outline text="SCRIPT Element"
              uri=".../scripts.html#h-18.2.1"/>
        <outline text-"Specifying the Scripting Language"
              uri-".../scripts html#h-1B.2.2">
          <outline text="Default Language"
              uri=".../scripts.html#h-18.2.2.1"/>
          <outline text-"Local Language Declaration"
              uri=".../scripts.html#h-18.2.2.2"/>
          <outline text-"References to HTML Elements"
              uri=".../scripts.html#h-18.2.2.3"/>
        </outline>

      </outline>

    </outline>
  </body>
</opml>
```

Обратите внимание на древовидную структуру документа OPML, узлы ветвления которого содержат в себе другие элементы, заключенные между открывающим и закрывающим тегами. В то же время «листья» этого дерева ничего в себя не включают.

Если вы размещаете OPML-документ на сервере с расширением **.opml,** обязательно нужно убедиться, что настройки сервера ставят этому расширение в соответствие тип содержимого **text/xml.** Аналогично, любые данные, размещенные на сервере в этом формате, должны указывать в заголовке тип содержимого **text/xml.**

XML-документ нельзя просто загрузить в окно или фрейм Internet Explorer и получить доступ к иерархии элементов документа. Обработчик, встроенный в IE, преобразует XML-документ в пригодный для просмотра (и печати) вид. В результате документ превратится в обычный документ HTML, засоренный всевозможной разметкой форматирования.

Для загрузки и чтения XML без дополнительной обработки IE и браузеры на основе Mozilla предоставляют отдельные виртуальные документы, которые не отображаются и, что более важно, сохраняют иерархию XML. В IE для этой цели используется элемент управления ActiveX **(Msxml.DOMDocument),** имеющийся

в настольных Windows-системах с пятой версии IE (но недоступный в IE 5.x для Mac). Mozilla следует стандарту W3C DOM, описывающему объект и метод для создания виртуальных документов такого рода (document.implementation.createDocument()). В этом рецепте применяется симметричный, работающий в разных браузерах подход к загрузке XML. Тег <object> в коде страницы загружает нужный элемент управления ActiveX. Если загрузка произошла успешно, то соответствующая IE ветвь кода в loadXMLDoc() готова к созданию контейнера документа, который будут использовать другие фрагменты сценария.

После того как специфичный для выбранной объектной модели виртуальный XML-документ будет создан, сценарий вызовет метод load(). К счастью, этот метод существует на обеих платформах (хотя не описан в W3C DOM Level2), причем принимает одинаковые аргументы и выполняет одинаковую работу. Затем с помощью вызова setTimeout() производится задержка, нужная, чтобы IE не начал обработку XML-данных до того, как они загружены.

При синтаксическом разборе иерархии XML-документа (это производится в функции drawOutline()) используется регулярность структуры элемента OPML документа body. Правда, в браузерах на основе Mozilla имеется один нюанс. Если документ OPML был передан с символами возврата каретки между строками, они будут восприняты как текстовые узлы. Чтобы обработать такую ситуацию, в коде функции drawOutline() имеется несколько мест, где выполнение цикла слегка меняется, если встретился узел с типом 3. Нас интересуют только элементы (узлы типа 1), содержащие текст и URL ссылки. Оставшаяся часть функции занята рекурсивными вызовами этой же функции, формирующими вложенные пункты меню, как это делается в рецепте 10.10.

Так как язык OPML имеет расширяемый набор атрибутов, можно добавлять в описание меню любую дополнительную информацию. Это может быть информация об изображении, если собираетесь использовать их в меню (URL, альтернативный текст и т. д.). Нужно не забывать использовать и головную секцию OPML-кода, которая может содержать важную для пользователя информацию, такую как даты, заголовки и начальное состояние пунктов меню.

Смотрите также

Формирование больших строк из отдельных фрагментов описывается в рецепте 1.1. Другие примеры использования setTimeout() можно увидеть в рецепте 4.5. Древовидное меню на основе данных JavaScript, более совместимое со старыми браузерами, можно увидеть в рецепте 10.10. Предварительная загрузка изображений показана в рецепте 12.1. В рецепте 12.7 показано, как управлять видимостью элементов.

11 Таблицы стилей

11.0. Вступление

Идея каскадных таблиц стилей (CSS) довольно проста: разделить содержимое страницы и правила его оформления. Сейчас, когда веб-разработчики все больше специализируются, эта технология позволяет писателям писать, а дизайнерам — оформлять, не мешая друг другу. Пожалуй, это упрощает и практическую сторону вопроса. Вместо того чтобы описывать оформление в атрибутах тегов, разбросанных по всему тексту, можно задать все значения в одном месте, после чего они будут автоматически применяться к содержимому страницы.

CSS является развивающимся стандартом. Вначале существовал CSS Level 1, частично поддерживаемый Internet Explorer 3, и более полно — Internet Explorer 4 и Navigator 4. Одно из расширений CSS, называемое CSS-Positioning, представлено как стандарт для описания точного положения элементов на странице (см. главу 13). CSS и CSS-P, а также много новых добавлений были объединены в CSS Level 2, в разной мере поддерживаемый браузерами, начиная с Netscape 6, IE 5 и Opera 5.

Присоединение стилей к документу

Существует три способа применить правила из таблицы стилей к документу:

- с помощью тега <style>;
- используя атрибут элемента style;
- импортировав данные из внешнего файла (см. рецепт 11.4).

При использовании тега <style> следует указывать MIME-тип используемого CSS-кода. Сейчас во всех таблицах стилей используется стандартный синтаксис, которому соответствует тип text/css. Так как устаревшие браузеры не распознают тег <style>, они выведут правила из таблицы стилей на экран, если не обернуть их тегами HTML-комментария. Таким образом, тег <style> должен иметь следующий формат:

```
<type="text/css">
<!-
одно или несколько правил CSS
->
</style>
```

Подобный синтаксис позволяет применять оформление, заданное в таблице стилей, к одному или нескольким элементам. Если нужно задать оформление только одного элемента, можно поместить описание его стиля в атрибут style его тега. Это описание представляет собой набор пар свойство стиля/значение свойства, формат которого несколько отличается от стандартного синтаксиса задания атрибутов HTML.

Некоторые авторы пользуются исключительно методикой на основе тега <style>, другие применяют комбинацию двух подходов. Первая методика более удобна при длительной поддержке, зато вторая удобнее при разовых изменениях. Но если вы рассчитываете применять к странице таблицы стилей большого объема, вам стоит обратиться к хранению описания стилей во внешнем файле (см. рецепт 11.4).

Синтаксис правил CSS

Определяя в теге <style> правило CSS, необходимо задать объекты, к которым оно будет применено, и описать свойства правила. Целевой элемент указывается с помощью селектора, которым может быть имя тега, значение атрибута class или значение атрибута id. Свойства стиля помещаются в фигурных скобках в соответствии со следующей схемой:

```
selector {свойство1:значение1; свойство2:значение2: ...}
```

Если селектором является имя тега, это имя остается без изменений:

```
h2 {свойство1:значение1; свойство2:значение2: ...}
```

Имена свойств в описании стиля нечувствительны к регистру символов (хотя в этой книге они всегда пишутся строчными буквами). Значения обычно не нужно обрамлять кавычками, кроме тех случаев, когда значение состоит более чем из одного слова. Для отделения имени свойства от значения служит двоеточие (и необязательный пробел). Если в правиле имеется более одного свойства, они должны быть разделены точкой с запятой.

Когда правила стиля назначаются через атрибут style в теге элемента, значение атрибута должно представлять собой список пар свойство/значение, но без фигурных скобок:

```
<h2 style="свойство1:значение1; свойство2:значение2: ...">...</h2>
```

Каждому свойству соответствует свой тип значений. Многие из них описывают физические размеры какого-либо рода, в значения таких свойств всегда следует помимо числового значения указывать единицу измерения. Например, если нужно установить размер шрифта в абзаце равным 14 пунктам, описание стиля должно выглядеть так:

```
P {font-size:14pt}
```

Значениями свойств часто являются предопределенные константы. Например, чтобы задать в абзаце курсивный шрифт, свойству font-style необходимо присвоить значение italic, как это показано ниже:

```
P {font-style:italic}
```

Если свойству может соответствовать более одного значения, эти значения должны быть разделены запятыми, хотя в некоторых случаях можно использо-

вать и пробелы. Например, много отдельных параметров шрифта можно описать с помощью свойства font, разделяя свойства пробелами:

```
p {font:12pt sans-serif bolder}
```

Браузер знает, какому из параметров шрифта какое значение соответствует, так как каждое из значений применимо только к одному из них.

Каскадность и специфичность

Применяя к странице правила из различных таблиц стилей, CSS-браузеры следуют строго определенным правилам. Нередко можно увидеть ситуацию, когда к одному элементу применяются несколько конфликтующих стилей. Указания каскадирования помогают браузеру понять, какой стиль применяется к каждому из элементов.

Одним из основных правил является тот факт, что чем строже правило описывает целевой элемент среди других элементов страницы, тем больший приоритет оно имеет. Например, если всем тегам <p> назначить один стиль с помощью тега <style> и, кроме того, одному из абзацев указать другой стиль через атрибут style, любые конфликты разрешаются в пользу второго определения. При этом свойства более широкого правила, не приводящие к конфликтам, применяются к элементу без изменения. Таким образом, глобальные правила наследуются элементами, но они могут быть переопределены.

11.1. Глобальные правила CSS

NN4 IE3

Задача

Необходимо применить определенное правило CSS ко всем элементам заданного типа на странице.

Решение

Нужно описать правило CSS в головной секции документа, указав в качестве селектора имя HTML-элемента:

```
<style type="text/css">
<!-
имяТега {свойство1:значение1; свойство2:значение2;  ..}
->
</style>
```

Здесь имяТега означает название тега без угловых скобок. Следующий пример устанавливает размер шрифта и высоту строк в 14 пунктов и 110 % для каждого элемента p:

```
<style type="text/css">
```

```
p {font-size:14pt; line-height:110%}
-->
</style>
```

Обсуждение

Тег <style> не распознается браузерами, предшествующими Navigator 4 или Internet Explorer 3, поэтому для нормального отображения страницы в старых браузерах следует скрыть определения правил с помощью тегов комментария HTML (иначе эти браузеры отобразили бы определения правил прямо на странице). Если же вы не заботитесь о старых программах, эти теги можно опустить.

Так как глобальные определения стилей применяются к каждому элементу заданного типа на странице, с их помощью можно задавать поведение элементов по умолчанию. Например, если вам не нравится, как в браузере выглядит текст, заключенный в теги (означающие выделение), можно переопределить поведение этого тега, чтобы оно отображало текст обычного вида, но красного цвета:

```
em {font-style:normal; color:red}
```

При этом на самой странице ни один из тегов менять не надо. Браузеры, не поддерживающие CSS, будут отображать содержимое тега без изменений, курсивом.

Смотрите также

В рецептах 11.2 и 11.3 показано, как уточнить область применения стиля, назначая его более узкому подмножеству элементов.

11.2. Назначение стиля подгруппе элементов

NN4 **IE4**

Задача

Разнородной группе элементов на странице необходимо назначить определенный стиль оформления.

Решение

Чтобы добиться нужного эффекта, в качестве селектора следует использовать имя класса или контекстный селектор. Если селектором является имя класса, то стиль будет применен ко всем элементам, атрибуту class которых присвоено

соответствующее имя. Имя такого селектора должно начинаться с точки, как в показанном ниже примере:

```
.hot {color:red: text-decoration:underline}
```

Но, задавая имя класса для элемента, точку указывать не нужно:

```
<p>А теперь кое-что <span class="hot">совершенно</span> другое</p>
```

Контекстный селектор позволяет определить внешний вид всех элементов одного вида, когда они размещены внутри некоторого другого тега. Например, следующее правило применяется ко всем тегам em, расположенным внутри какого-либо элемента p:

```
p em {font-size:16pt: font-style:normal}
```

Если же элемент em будет расположен, скажем, внутри элемента li, правило к нему применено не будет.

Обсуждение

Селекторы классов и контекстные селекторы являются довольно мощными инструментами CSS. Например, можно применить правило к заданной группе одинаковых тегов, например, так:

```
p.narrow {margin-left:5em; margin-right:5em}
```

Это правило будет применено ко всем элементам p, атрибуту class которых присвоено имя narrow:

```
<p class="narrow">...</p>
```

Затем можно определить селектор класса, относящийся к элементам div, атрибут class которых содержит значение narrow:

```
div.narrow {margin-left:7em; margin-right:7em}
```

Можно еще больше уточнить контекстный селектор, описывая столько уровней вложения, сколько нужно. В следующем примере определение стиля применяется к элементам span, вложенным в тег em, который, в свою очередь, вложен в элемент p с классом narrow:

```
p.narrow em span {background-color:yellow}
```

Одной из сильных сторон CSS является то, что это правило унаследует прочие стили, указанные для элементов p.narrow и em.

Более того, используя контекстные селекторы, можно переопределить значения уже описанных свойств. Далее следует пример, в котором для элемента em, расположенного внутри тега p, переопределяется значение одного из атрибутов:

```
p em {font-size:16pt; font-style:normal}:
p em span {font-size:18pt}

<p>Весь <span>этот ТеКСТ</span> размером <em> 16pt  за исключением <span> этого фрагмента
высотой 18</span></em></p>
```

Элемент span внутри тега p наследует значение font-style, но устанавливает собственный размер шрифта.

Имена классов (также называемые идентификаторами) имеют только несколько ограничений:

- должны быть одним словом (не должны содержать пробелы);
- не должны содержать знаков препинания (кроме символа подчеркивания);
- первый символ не должен быть цифрой;
- это не должно быть одним из зарезервированных слов ECMAScript (см. приложение B);
- не все из этих ограничений критичны для CSS, но если вы хотите обращаться к объектам из сценария, несоблюдение этих правил может вызвать проблемы.

Смотрите также

Как задать правило для одного элемента, показано в рецепте 11.3. Переопределение правила для одного элемента описано в рецепте 11.9.

11.3. Задание правила для одного элемента

NN4 **IE4**

Задача

Необходимо выделить один элемент, назначив ему отдельное правило из таблицы стилей.

Решение

В качестве селектора для CSS-правила можно использовать идентификатор элемента (значение атрибута id). Этот идентификатор должен быть предварен знаком #, как в показанном ниже примере:

```
#special {border:5px; border-style:ridge; border-col or: red}
```

Применить это правило к элементу можно так:

```
<div id="special">...</div>
```

При написании HTML-кода следует назначить такой идентификатор не более чем одному элементу на странице. Повторяющиеся идентификаторы могут нарушить работу сценария.

Обсуждение

Вы можете определять идентификаторы элементов полностью на свое усмотрение, но следуя тем же ограничениям, что относятся к именам классов Следует

учитывать, что если у элемента заданы как идентификатор, так и класс, правила, заданные для идентификатора, имеют приоритет над правилами для класса в случае конфликта. Конфликт возникает только в том случае, если и для класса, и для идентификатора в таблице стилей задано значение одного и того же свойства.

Смотрите также

Селекторы классов описываются в рецепте 11.2. Перегрузка описаний стилей показана в рецепте 11.9. Соглашения об именовании идентификаторов, относящиеся и к значениям атрибута id, приведены в рецепте 4.1.

11.4. Внешние таблицы стилей

NN4 **IE4**

Задача

Следует применить одну и ту же стратегию оформления к нескольким страницам сайта, не встраивая CSS-код в каждую страницу.

Решение

Таблицу стилей можно задать в виде отдельного документа **.css**, который импортируется в исходный с помощью тега <link>:

```
<link rel="stylesheet" type="text/css" href="myStyleSheet.css">
```

Файл описания таблицы стилей должен содержать только правила CSS. Теги **<style>** или HTML-комментарии в него включать нельзя.

Обсуждение

В этом решении показан полностью совместимый пример, работающий во всех поддерживающих CSS браузерах. Наиболее современные браузеры, соответствующие спецификациям CSS2, могут в дополнение к показанному использовать другой синтаксис внутри тега **<style>**:

```
<style type="text/css">
<!-
@import url(myStyleShees.css)
->
</style>
```

Такую запись можно использовать в браузерах, начиная с Internet Explorer 4 и Netscape 6. Некоторые разработчики используют такую запись, чтобы исключить Navigator 4, который просто игнорирует ее. Правило **@import** можно комбинировать с другими правилами, заданными в той же паре тегов <style>.

Файл, содержащий описание таблицы стилей, должен иметь расширение .css. Кроме того, чтобы Netscape 6 и более старшие версии работали нормально (чтобы просто соответствовать стандартам), сервер должен ставить в соответствие этому расширению MIME тип text/css.

Смотрите также

Подключение таблиц стилей, адаптированных к определенному браузеру и операционной системе, смотрите в рецепте 11.5. Изменение импортированной таблицы стилей после того, как страница уже загружена, демонстрируется в рецепте 11.6. Импорт содержимого HTML демонстрируется в рецепте 14.3.

11.5. Использование ОС- и браузер-специфичных таблиц стилеи

NN4 IE4

Задача

Необходимо, чтобы на разных компьютерах к странице применялось разное оформление.

Решение

В головной секции страницы можно поместить JavaScript код, который определяет операционную систему и браузер и вписывает в страницу соответствующий тег <link>. В следующем примере пользователи Mac и все остальные пользователи получают разное оформление:

```
<head>

<script language="JavaScript" type="text/javascript">
var cssFile = (navigator.userAgent.indexOf("Mac") != -1) ?
    "styles/macCSS.css" : "styles/pcCSS.css";
document.write("<link rel='stylesheet' type='text/css' href-'" + cssFile + "'>");
</script>

</head>
```

Такой способ подключения таблиц стилей можно комбинировать с подключением стилей с помощью статичных тегов <link> на той же странице или с другими способами подключения.

Если ваш код должен соответствовать строгой версии XHTML, необходимо, чтобы и динамически генерируемый код соответствовал им. Поэтому оператор document.write() следует переписать так:

```
document.write("<link rel='stylesheet' type='text/css' href='" +
    cssFile+"'  />");
```

Чтобы ваш сценарий более строго соответствовал стандарту XHTML, можно переместить загрузку таблицы стилей во внешнюю библиотеку .js.

Обсуждение

Использование разных таблиц стилей для разных браузеров приводит к тем же проблемам, что и применение нескольких версий страницы. Все изменения в дизайне необходимо делать во всех версиях и тщательно тестировать на указанных платформах. Кроме того, можно управлять обработкой CSS-правил в некоторых версиях Internet Explorer для Windows, которые не полностью совместимы со стандартом (см. рецепт 11.13).

Смотрите также

Как управлять режимом совместимости с CSS в IE 6 для Windows, показано в рецепте 11.13. Определить операционную систему и версию браузера поможет рецепт 5.5.

11.6. Изменение импортированных стилей после загрузки

NN6 **IE5(Win)**

Задача

Необходимо предоставить пользователям возможность выбирать один из нескольких вариантов оформления сайта («skin»), подгружая к уже открытой в браузере странице различные .css-файлы.

Решение

Если вы приняли идею разделения содержимого и оформления, внешний вид ваших страниц определяется исключительно таблицами стилей. Разработав новую таблицу стилей, можно полностью изменить вид сайта: цвета фона и текста, отступы и поля, шрифты и т. д., не трогая содержимое. С помощью тега <link> можно установить обычное оформление для страницы, а затем изменить URL в нем, чтобы использовать другие варианты оформления. Вот пример элемента <link>:

```
<link id="basicStyle" rel="stylesheet" type="text/css"
   href="styles.normal.css" />
```

Чтобы загрузить в страницу новую таблицу стилей, можно использовать следующий код:

```
document.getElementById("basicStyle").href = "styles/crazySkin.css";
```

После того как новое оформление будет загружено, Internet Explorer 5 для Windows и Netscape 6 автоматически применят его к странице. IE 5 для Macintosh автоматически новое оформление не применяет.

Обсуждение

Существуют и другие способы обращаться к таблицам стилей, например с помощью коллекции document.styleSheets. Но Netscape младше версии 7 реагирует неправильно, если изменять значение свойства href у объекта, адресованного таким способом. С другой стороны, если необходимо поддерживать Internet Explorer 4 (который не поддерживает метод W3C DOM getElementById()), придется использовать либо коллекцию document.styleSh.eets, либо имеющуюся только в IE коллекцию document.all.

Если вы предоставляете пользователям возможность управлять внешним видом сайта, предлагая несколько вариантов оформления, нужно сохранять пользовательские настройки, чтобы автоматически восстановить их при следующем посещении. Наиболее удобный способ — сохранять настройки на машине клиента в cookie. Для примера, пусть на странице имеется группа переключателей, каждый из которых определяет один из вариантов оформления. В атрибуте value каждой кнопки должен храниться URL соответствующей таблицы стилей. Обработчик события onclick у кнопки должен применить выбранное пользователем оформление и сохранить выбор в cookie (например, с помощью утилит из рецепта 1.9):

```
function setSkin(evt) {
    // обработка моделей события IE и W3C
    evt - (evt) ? evt   ((window.event) ? window.event : null):
    if (evt) {
        var btn - (evt.srcElement) ? evt.srcElement . evt.target;
        document.getElementById("basicStyle").href = btn.value;
        setCookieValue("skin", escape(btn.value));
```

Кроме того, необходим сценарий, который возьмет на себя ответственность за применение сохраненной таблицы стилей при следующей загрузке страницы. Но следует учитывать возможность того, что либо сценарии, либо cookie в браузере будут отключены. Поэтому требуется поместить на страницу жестко заданный элемент <link>, за которым последует сценарий, считывающий сохраненные настройки и помещающий нужную ссылку в свойство href:

```
var skin = getCookieValue("skin");
if (skin) {
    document.getElementById("basicStyle").href - unescape(skin);
}
```

Наконец, в завершение нужно установить переключатель в состояние, соответствующее загруженным настройкам. Для этого можно либо использовать document.write(), чтобы сгенерировать HTML-код переключателя, либо в обработчике события onload перебрать кнопки на странице и найти ту, значение которой совпадает с URL загруженной таблицы.

Смотрите также

Сценарий, переключающий уже загруженные таблицы стилей, показан в рецепте 11.8. Работа с cookie подробнее описана в рецепте 1.9. В рецепте 12.4 показано, как использовать cookie для установки стилей.

11.7. Включение и отключение таблиц стилей

NN6 IE4

Задача

Чтобы динамически изменять оформление страницы, необходимо активизировать и деактивизировать таблицы стилей.

Решение

Для отключения таблицы стилей следует присвоить булевскому свойству disabled объекта stylesheet значение true:

```
document.styleSheets[1].disabled = true;
```

Наоборот, чтобы опять включить таблицу стилей, необходимо присвоить этому свойству false:

```
document.styleSheets[1].disabled = false;
```

Однако даже некоторые современные браузеры не всегда правильно реагируют на такой оператор. Как ни странно, можно включать и выключать элемент link, загрузивший таблицу стилей. У элемента style есть свойство disabled, поддерживаемое IE 4 и старше, а также NN 6 и старше.

Обсуждение

Возможность включать и выключать таблицы стилей дает еще один способ реализации настраиваемого интерфейса для страницы. Для этого на странице должно быть несколько тегов <style>, каждый из которых содержит полное описание одного из вариантов оформления. Переключатель может отключать все стили, кроме одного, как это делается в рецепте 12.4.

Смотрите также

Загрузка таблиц стилей «на лету» показывается в рецепте 11.6. В рецепте 11.8 показано, как переключаться между уже загруженными таблицами стилей. Как сохранять между посещениями настройки оформления в cookie, описано в рецепте 12.4.

11.8. Смена стиля элемента

NN6 IE4

Задача

В ответ на действия пользователя необходимо применять к элементу разные таблицы стилей (например, в ответ на перемещение мыши или на щелчок на элементе).

Решение

Прежде всего следует задать два определения стиля с разными селекторами класса. Затем нужно написать обработчик события, который бы менял значение свойства **className** у элемента:

```
<style type="text/css">
.unhilited {background-color:white}
.hilited {background-color:yellow; text-decoration:underline}
</style>

<script type="text/javascript">
function setHilite(evt) {
    evt = (evt) ? evt : ((window.event) ? window.event : null);
    if (evt) {
        var elem = (evt.srcElement) ? evt.srcElement : evt.target;
        elem.className = "hilited";
    }
}
function setUnHilite(evt) {
    evt = (evt) ? evt : ((window.event) ? window.event : null);
    if (evt) {
        var elem = (evt.srcElement) ? evt.srcElement : evt.target;
        elem.className = "unhilited";
    }
}

<span class="unhilited" onmouseover="setHilite(event)"
    onmouseout="setUnHilite(event)">Какой-то текст.</span>
```

Работа со свойством dassName, как это показано в примере, — более надежный (для ранних версий Netscape 6) подход, чем манипулирование объектами stylesheet. Пожалуй, это наиболее широко используемый и поддерживаемый способ реализации динамических стилей.

Обсуждение

Если вы меняете стиль только для одного элемента, можете попробовать использовать вместо имени класса и селектора класса идентификатор элемента и, соответственно, селектор идентификатора. Однако атрибут id любого элемента не должен меняться без крайней на то необходимости.

Когда сценарий назначает новое описание стиля элементу, оно не наследует ни одно из старых свойств. В предыдущем примере селектор класса hilighted устанавливает свойству text-decoration значение underline, после того как элементу присваивается описание класса unhilited, автоматически восстанавливается исходное значение этого свойства, которое элемент унаследовал из таблицы стилей по умолчанию.

Смотрите также

Как включать и выключать таблицы стилей, показано в рецепте 11.7.

11.9. Подмена правила таблицы стилей

NN4 IE4

Задача

Отдельный элемент должен использовать глобальное правило, за исключением одного или двух свойств, значение которых уникальны для этого элемента.

Решение

Есть два общих подхода к решению этой задачи. В первом подходе создается описание стиля с идентификатором элемента в качестве селектора. У элемента должны быть заданы значения как атрибута id, так и class. Тогда к нему вначале применяется правило класса, а затем правило, связанное с идентификатором, может подменить значения некоторых свойств. Вот пример двух описаний стиля и элемента, к которому они применены:

```
p.narrow {font-size:14pt; margin-left:2em; margin-right:2em}
#narrow_special {margin-left:2.5em; margin-right:2.5em;
                 border:5px ridge red}

<p class="narrow" id="narrow_special">...</p>
```

Другой подход состоит в том, чтобы устанавливать уникальные для элемента настройки стиля с помощью атрибута style в его теге. Ниже показан пример, эквивалентный предыдущему:

```
p.narrow {font-size:14pt; margin-left:2em; margin-right:2em}

<p class=narrow
   style=" margin-left:2.5em; margin-right:2.5em; border:5px ridge red ">
...</p>
```

Обсуждение

Правила каскадирования уступают преимущество правилам, присвоенным отдельному элементу. Наибольший приоритет имеет описание, помещенное в атрибуте

style тега элемента. Таким образом, даже если атрибуту style присвоено значение, прочие правила все равно влияют на те описания элемента, которые не были назначены в атрибуте.

Смотрите также

Базовые сведения о назначении правил из таблиц стилей приведены в рецептах 11.1-11.3.

11.10. Создание оформленного элемента

NN4 IE3

Задача

Следует назначить определенный стиль секции содержимого документа, до этого не выделенной никакими тегами.

Решение

Необходимо завернуть фрагмент текста в пару тегов, или , и назначить стиль через тип элемента, класс или идентификатор:

```
<p>А теперь кое-что <span class="hot"> совершенно</span> иное</p>
```

Обсуждение

Несмотря на то что в HTML 4.0 описано множество контекстных тегов (таких как <address> или <blockquote>), их имена не всегда соответствуют контексту документа. Хотя можно использовать XML-методики, чтобы заполнить этот пробел (разрабатывая собственные теги), для задания контекста можно применять и HTML.

Для оформления фрагмента текста внутри абзаца обычно используется элемент span. Элемент div автоматически задает блок, то есть содержимое div, как и текст после него, начинается с новой строки. Этот элемент часто используется, если нужно сменить выравнивание, скажем, с левого на выравнивание по центру. Кроме того, div является подходящим контейнером для позиционируемого содержимого страницы.

Смотрите также

Как использовать div- и CSS-позиционирование, показано в рецептах главы 13. Преобразование выделения в элемент, пригодный для оформления, демонстрируется в рецепте 15.2.

11.11. Создание выровненных по центру элементов

NN4 IE4

Задача

Абзац или другой элемент документа фиксированной ширины должны быть выровнены по центру страницы.

Решение

Можно использовать два подхода. Один из них обладает обратной совместимостью, в то время как второй работает только в новейших браузерах в режиме совместимости со стандартом (см. рецепт 11.13). Суть первого подхода в том, чтобы заключить элемент в подходящий элемент div, а затем указать во внешнем элементе привязку текста по центру. Например, чтобы расположить по центру страницы группу абзацев уменьшенной ширины, стиль абзацев можно описать так:

```
p.narrow {width:70%}
```

Затем абзацы нужно завернуть в элемент div, описание стиля которого гласит:

```
div.centered {text-align:center}
```

Если браузер совместим со стандартами, можно обойтись без дополнительного div, указав для абзаца значения левого и правого полей:

```
p.narrow {width:70%; margin-left:auto; margin-right:auto}
```

Браузеры в старом режиме (в том числе IE 6) игнорируют установки полей, и такие абзацы будут выровнены по левому краю.

Обсуждение

Эти методы вытесняют другие технологии, не рекомендованные к использованию в HTML 4: элемент center и атрибут align у других элементов. Вертикальное выравнивание элемента по центру — непростая задача, для решения которой обычно нужны абсолютно позиционируемые элементы, описанные в главе 13.

Смотрите также

Режимы совместимости IE 6 смотрите в рецепте 11.13. Как расположить позиционируемый элемент по центру окна или фрейма, рассказано в рецепте 13.7.

11.12. Определение эффективных значений таблицы стилей

NN6 **IE5**

Задача

Сценарий должен выяснить значение одного из свойств таблицы стилей, изначально установленное с помощью тега **<style>** или импортированной таблицы стилей.

Решение

Ниже показана функция **getElementStyle()**, работающая в браузерах, поддерживающих стандартный синтаксис обращения к элементам W3C DOM, а также в браузерах, поддерживающих объект IE currentStyle или метод W3C DOM **window.getComputedStyle()**:

```
function getElementStyle(elemID, IEStyleProp, CSSStyleProp) {
    var elan = document.getElementById(elemID);
    if (elem.currentStyle) {
        return elem.currentStyle[IEStyleProp];
    } else if (window.getComputedStyle) {
        var compStyle = window.getComputedStyle(elem, "");
        return compStyle.getPropertyValue(CSSStyleProp);
    }
    return ""
}
```

Эта функция возвращает значение свойства, используемое браузером для задания стиля отображения элемента. В качестве параметра функции нужно передать имя свойства (в формате IE или CSS).

Обсуждение

Обычно можно считать, что определяя значение стиля элемента, вы просто считываете значение его свойства **style.имяСвойства**. Но это справедливо, только если стиль элемента указан через атрибут style или предварительно был изменен из сценария. Так как чаще стиль элемента определяется отдельно (с помощью пары тегов **<style>** или импортируется тегами <link> или правилом @import), такой простой метод не работает. Полученное значение будет пустой строкой, даже если элементу задан соответствующий стиль.

Определение удаленно заданных настроек стиля требует использования объектной модели документа браузера. Модель Internet Explorer включает в себя свойство элемента, называемое currentStyle. Этот объект содержит большую часть свойств, имеющихся в свойстве style (хотя не все), значения которых соответст-

вуют действительному стилю, примененному к элементу. Свойства этого объекта доступны только для чтения. Таким способом можно прочитать даже значения по умолчанию, взятые из собственной таблицы стилей браузера.

В браузерах, основанных на Mozilla, имеется метод W3C DOM объекта window getComputedState(), возвращающий объект, свойства которого подобны свойствам style. Тем не менее этот метод следует использовать в два шага: сначала получить объект style (обычно это объект W3C DOM CSSStyleDeclaration), затем вызвать у этого объекта метод getPropertyValue().

Два подхода зачастую требуют различных способов обращения к свойствам стилей, как будто двух разных моделей доступа недостаточно. В случае объекта IE currentStyle, необходимо ссылаться по тем же именам свойств объектной модели, что используются для установки и считывания свойств стилей. Поэтому для обращения к свойствам, имена которых содержат дефис, следует удалять дефисы и делать первую после дефиса букву заглавной (то есть имя margin-left станет **marginLeft**). Но имена свойств для метода **getPropertyValue()** должны быть в формате CSS (то есть margin-left остается margin-left), поэтому показанная в примере функция getPropertyValue() требует передачи двух параметров. Один из них — это имя для IE, второй — имя в формате W3C DOM. Например, чтобы определить цвет заднего фона для элемента **myDiv**, вызов должен выглядеть так:

```
var divColor = getElementStyle("myDiv", "backgroundColor",
   "background-color");
```

Также следует учитывать, что разные браузеры возвращают значения разного типа для разных свойств, в особенности это касается цветов, указанных не с помощью синтаксиса rgb(r,g,b). Например, если указать цвет обычным текстовым именем (например, orange), значения, возвращаемые разными браузерами, будут в разном формате. Если же указывать цвет с помощью rgb(r,g,b), это решит проблему в большинстве браузеров (кроме Netscape 6.2).

Значения свойств CSS, содержащие размеры, часто включают единицу измерения (пикселы, пункты и т. д.). Поэтому если вы рассчитываете использовать значение свойства для математических вычислений, например, чтобы добавить пять пикселов к правой границе элемента, необходимо извлечь числовую часть такого значения. Для таких значений можно использовать функции **parseInt()** для целых и parseFloat() для действительных чисел, так как они допускают наличие букв после числа (например, 0.5ет).

Если установить значение свойства с помощью принадлежащего элементу объекта style, то затем его значение можно считывать из этого свойства. Но для большей уверенности можно продолжать использовать getElementStyle(), так как эта функция возвращает эффективное значение стиля элемента.

Смотрите также

Смотрите рецепты 9.3, 9.9, 13.12 и 13.13, в которых используется функция **getElementStyle()**.

11.13. Перевод браузеров версии 6 в режим совместимости со стандартами

NN6 IE6

Задача

Браузеры IE 6 и NN 6 можно перевести в режим, в котором они будут придерживаться W3C стандартов CSS. Прежнее поведение сейчас представляется нестандартным.

Решение

Первым элементом любого документа, соответствующего современным стандартам, нужно поставить элемент DOCTYPE. В некоторых случаях, чтобы перевести браузер в стандартный режим, требуется указывать URL.

Если документ в целом следует рекомендациям W3C HTML 4.0, но может включать не рекомендованные элементы из более ранних спецификаций, нужно использовать следующий элемент (для режима совместимости необходимо указать URL):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
        "http://www.w3.org/TR/REC-html40/loose.dtd">
```

Для документа, задающего структуру фреймов и следующего рекомендациям HTML 4.0, используйте следующий элемент (для режима совместимости нужно указать URL):

```
<!DDCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
        "http://www.w3.org/TR/REC-html40/frameset.dtd">
```

Для документа, строго соответствующего рекомендациям, необходимо взять запись (для режима совместимости URL указывать не нужно):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
        "http://www.w3.org/TR/REC-html40/strict.dtd">
```

Если разметка документа в целом следует рекомендациям W3C XHTML 1.0, но может включать нестандартные элементы, применяйте следующую запись (для режима совместимости URL указывать не нужно):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Задающий структуру фреймов документ с добавочными терминами, относящимися к фреймам, и следующий рекомендациям W3C XHTML 1.0 должен начинаться так (для режима совместимости URL указывать не нужно):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Документ, строго соответствующий рекомендациям W3C XHTML 1.0, должен начинаться так (для режима совместимости URL указывать не нужно):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Наконец, следующая запись соответствует документу, разметка которого строго следует рекомендациям W3C XHTML 1.0 (для режима совместимости URL указывать не нужно):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
       "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Элемент DOCTYPE для определенного уровня HTML или XHTML, добавленный в документ, не влияет на способность браузера распознавать и отображать теги и атрибуты, не описанные рекомендациями.

Обсуждение

Различия между режимом совместимости со стандартами и старым режимом работы (иногда называемым «капризным») зависят от браузера. В Netscape 6 и позже вы можете даже не заметить разницу между этими режимами, различия проявляются лишь в незначительных изменениях числа пикселов между элементами управления. В IE 6 различия более значительны. W3C адаптировал способы измерения различных размеров элементов, таких как границы, пробелы и поля, иначе, чем это сделано в Internet Explorer 4. По многим причинам в IE 6 режим совместимости со стандартами более предсказуем в том, как он реагирует на различные размеры, границы и поля других элементов. Результаты работы браузера в таком режиме более соответствуют тем, что получаются в Netscape 6 при тех же спецификациях DOCTYPE. Создавая страницы, следует использовать режим совместимости, чтобы выработать у себя привычку работать в соответствии со спецификациями W3C (или, по крайней мере, в соответствии с тем, как браузеры интерпретируют их). Более старые браузеры игнорируют декларацию DOCTYPE.

В IE 6 и NN 7 имеется доступное для сценария свойство document.compatMode, позволяющее определить режим работы браузера. Это свойство может принимать одно из значений BackCompat или CSS1Compat.

Теоретически в XTML-странице должна быть декларация xml вместе с информацией о кодировке текста:

```
<? xml version="1.0" encoding="UTF-8" ?>
```

Но IE 6, встретив этот тег, начинает работать в режиме обратной совместимости, несмотря на декларацию DOCTYPE. Чтобы пройти проверку на соответствие XHTML, следует указать информацию о кодировке в элементе meta в головной секции документа:

```
<meta http-equiv="Content-Type" content="text/html; charset-utf-8" />
```

Смотрите также

Примеры того, как сценарии и таблицы стилей часто должны учитывать различные модели CSS в IE 6, вы можете увидеть в рецептах 9.3, 11.11, 13.6, 13.7, 13.11, 13.13, 15.5 и 15.10.

Визуальные эффекты для статичных элементов

12.0. Вступление

Динамическая часть DHTML не ограничивается порхающими по странице элементами, выпрыгивающими из ниоткуда иерархическими меню и перетаскиваемыми по странице объектами. Динамическим может быть и элемент, который за все время своей жизни на странице не сдвинется ни на пиксел, поскольку можно динамически менять его свойства, влияющие на вид его содержимого. Подобные изменения могут происходить автоматически или в ответ на действия пользователя.

Обращение к объектам страницы

Чтобы иметь возможность изменить характеристики элемента на странице, сценарий должен быть способен «общаться» с ним. На заре развития клиентских сценариев только небольшая часть элементов была доступна в виде объектов. Это были наиболее интерактивные из них, такие как элементы форм (кнопки, поля ввода и др.). Для получения ссылки на эти элементы приходилось использовать синтаксис, отвечающий иерархии элементов: начиная с объекта window, а затем последовательно фокусируясь на конкретном элементе. Поскольку под объектом window обычно предполагалось текущее окно, ссылки начинались с объекта document. Например, если с помощью атрибута name назначить идентификаторы для элементов a, form и input, эти идентификаторы можно было использовать при построении ссылки:

```
document.имяСсылки
document.имяФормы
document.имяФормы.имяЭлементаУправления
```

Если документ содержит несколько типов доступных элементов, элементы одного типа группируются в массивы (коллекции), которые также можно использовать для обращения к ним. В качестве индекса в таком массиве может выступать либо целое число, либо строка, содержащая значение атрибута name:

```
document.links[i]
document.forms["имяФормы"]
document.forms[2].elements["имяЭлемента"]
```

Эти старомодные способы обращения поддерживаются современными браузерами, чтобы обеспечить работу громадного объема уже существующего кода, в котором применяются подобные методики.

Первым браузером, который в своей объектной модели предоставил доступ ко всем элементам, стал Internet Explorer 4. В нем сценарии могут обращаться ко всем элементам, у которых задан атрибут name, или, что предпочтительнее, — id. Самым распространенным способом обращения в IE 4 является использование коллекции **document.all**, содержащей ссылки на все элементы документа, вне зависимости от их вложения и расположения на странице. Реализовано несколько вариантов синтаксиса, которые можно использовать для обращения к элементам с помощью этой коллекции:

```
document.all.идентификаторЭлемента
document.all["идентификаторЭлемента"]
document.all.("идентификаторЭлемента")
```

Последние два варианта особенно удобны при создании многоцелевой функции, получающей идентификатор элемента, с которым нужно производить действия, в виде строки. Более того, объектная модель IE позволяет полностью опустить имя коллекции и обращаться просто по идентификатору:

```
идентификаторЭлемента
```

С целью стандартизации различных D0M, чтобы объединить документы HTML и XML, были приняты рекомендации World Wide Web Consortium (W3C). Рабочая группа W3C DOM решила разработать собственную модель и синтаксис обращения к элементам. Объект document остался на месте, но обращение к элементам по строковому идентификатору производится с помощью нового метода:

```
document.getElementById("идентификаторЭлемента")
```

Этот стандартизованный синтаксис поддерживается IE 5 и Netscape 6, а также более поздними версиями этих браузеров. Таким образом, на сегодняшний день он поддерживается большинством браузеров. Исключениями являются организации, где в качестве внутреннего стандарта все еще принят Netscape 4, а также небольшие группы пользователей IE 4 и малочисленные группы пользователей более старых браузеров.

Чтобы написать код, использующий такие возможности, как динамические стили, следует устранить неравенство между documentall и document.getElementById(), а также предотвратить выполнение кода в менее современных браузерах, которые не знают, что с ним делать. Ниже показан шаблон, демонстрирующий, как обратиться к элементу, идентификатор которого задан в виде строки:

```
function myFunction(elemID) {
    var elem - (document.getElementById) ? document.getElementById("elemID")
              :((document.all) ? document.all("elemID") · null):
    1f (elem) {
        // Выполнение действий над элементом
    }
}
```

Здесь отдается предпочтение синтаксису W3C DOM, если поддерживается несколько вариантов (как, например, в IE 5). Кроме того, обратите внимание,

как в устаревших браузерах, неспособных обратиться к нужному элементу, пропускается код функции.

Но если ваша функция работает только с элементами, которые доступны для сценариев с самого начала, вам не обязательно использовать все эти сложные конструкции. В такой функции можно применять и устаревший синтаксис, даже если остальные части вашего кода построены с помощью более новых методик. Например, обращаться к изображениям позволяли все поколения браузеров, кроме самых первых, которые легко отсеять (см. рецепт 12.1).

В рецептах из этой книги вы везде будете видеть синтаксис W3C DOM, кроме следующих случаев:

- в рецепте приводится код, предназначенный для работы с различными объектными моделями;
- используемые элементы поддерживаются браузерами, предшествующими IE 4 и NN 6.

Если же вы намереваетесь использовать рецепт, ориентированный на W3C DOM в более разнообразном окружении, следует применять методику определения объектов, показанную ранее в myFunction().

Обращение к элементам из событий

Если некоторая функция определена обработчиком события, в ней почти наверняка потребуется получить ссылку на источник этого события. Internet Explorer и W3C DOM предлагают разные модели событий и синтаксис, но эти варианты достаточно похожи, благодаря чему можно привести их к одному виду. В следующей функции показано, как получить для дальнейшей обработки ссылку на целевой элемент события:

```
function myEventFunction(evt) {
    evt = (evt) ? evt : ((window.event) ? window.event : null);
    if (evt) {
        var elem = (evt.target) ? evt.target :
                   ((evt.srcElement) ? evt.srcElement : null);
        if (elem) {
            // Выполнение действий над элементом
        }
    }
}
```

Для более подробного обсуждения обработки событий в несовместимых объектных моделях смотрите главу 9.

Стиль элемента

Вы всегда можете рассчитывать на то, что в браузере, поддерживающем обращение к отдельным элементам HTML всех типов, каждый элемент имеет свойство style. В нем содержится ссылка на объект свойства которого совпадают с именами CSS-свойств стиля элемента (иногда имена немного изменены, чтобы соот-

ветствовать синтаксису JavaScript). Ссылка на одно из значений стиля элемента должна следовать такому синтаксису:

```
ссылкаНаЭлемент.style имяСвойстваСтиля
```

Например, показанное далее выражение назначает новое значение свойству color элемента, используя для обращения к элементу запись W3C DOM:

```
document.getElementById("mainHeading").style.color = "#ffff00";
```

Версия, предназначенная только для IE, полностью идентична, за исключением методик обращения к элементу:

```
document.all.mainHeading.style.color = "#ffff00";
```

Если имя CSS-свойства содержит тире, в соответствующем свойстве JavaScript тире выбрасывается, а первая буква следующего за тире слова переводится в верхний регистр. Это позволяет использовать имя свойства в языке JavaScript (а также в других). Например, следующая строка кода присваивает новое значение свойству font-size:

```
document.getElementById("link12").style.fontSize = "#ffff00";
```

Изменение значения CSS-свойства для элемента — сравнительно простая задача (есть несколько способов, показанных в рецептах 11.7 и 11.8). Но задача определения текущего значения уже сложнее, особенно если его значение установлено не из атрибута style элемента. Более подробно этот вопрос рассматривается в рецепте 11.13.

12.1. Предварительная загрузка изображений

NN3 IE4

Задача

Необходимо, чтобы меняющееся при наведении мыши изображение менялось мгновенно даже в первый раз, не заставляя пользователя ждать, пока загрузится с сервера новое.

Решение

Следует начать с создания двух объектов, имена свойств которых соответствуют идентификаторам изображений. Затем свойства одного объекта будут содержать ссылки на обычные изображения, а свойства другого — ссылки на подсвеченные:

```
if (document.images) {
    var imagesNormal = new Object();
    imagesNormal["home"] = new Image(20, 50);
    imagesNormal["home"].src = "img/homeNormal.jpg";
```

```
imagesNo:mal["products"] - new Image(20. 50):
imagesNormal["products"].src  - "img/prodNormal.jpg":

var imagesHilite = new Object():
imagesHilite["home"] = new Image(20. 50):
imagesHilite["home"].src - "img/homeHilite.jpg":
imagesHilite["products"] - new Image(20. 50):
imagesHilite["products"].src  = "img/prodHilite.jpg":
```

После того как страница загрузится (смотрите замечания в обсуждении), нужные изображения будут загружены в кэш браузера, но показаны не будут. В рецепте 12.2 показано, как можно использовать загруженные изображения.

Обсуждение

Хотя сейчас осталось очень мало браузеров, неспособных менять изображения, перед выполнением этого кода все равно желательно выполнять проверку, чтобы удостовериться в существовании массива document. images. Поскольку предварительная загрузка не включает в себя отображение, другие проверки здесь не нужны.

Использование объектов для хранения ссылок не является необходимым для предварительной загрузки. Тем не менее такие объекты могут упростить управление активными изображениями, как это демонстрируется в рецепте 12.2. Совпадение отдельных свойств объекта с идентификаторами элементов img здесь не случайно.

Каждое свойство содержит ссылку на объект Image. Данный объект существует только в памяти и никогда не отображается на экране. Но этот объект имеет те же свойства, что и элемент img, в частности размеры изображения (height и width), а также URI (src). Передавать в конструктор Image() размеры изображения не обязательно, но это может слегка ускорить загрузку, так как браузеру не придется их рассчитывать.

Основная работа по предварительной загрузке происходит в тот момент, когда свойству src присваивается URI. При этом браузер немедленно разрешает URI и пытается загрузить изображение.

Если вы столкнулись с ситуацией, когда предварительная загрузка не работает (то есть наблюдается заметная задержка при смене изображения), потенциальной причиной проблемы может стать неправильная настройка сервера. Не так уж редко некоторые веб-серверы (особенно те из них, которые поддерживают часто обновляющееся содержимое) для некоторых или даже для всех MIME-типов посылают HTTP-заголовки, предотвращающие кэширование данных. Убедитесь, что HTTP-заголовки ваших изображений не содержат срок действия или инструкцию «no-cache».

Кроме того, пользователи также могут помешать предварительно загрузить изображения, отключив кэш в настройках браузера. Вы не сможете отменить эту настройку, но такие пользователи должны видеть полную загрузку всего содержимого страницы, независимо от того, как часто они ее посещают.

Многие браузеры позволяют проверить, находятся ли ваши изображения в кэше. В IE для Windows откройте окно Сервис • Свойства обозревателя (Tools • Internet Options). На вкладке Общие (General) щелкните на кнопке Настройка (Settings) в секции Временные файлы Интернета (Temporary Internet Files). Далее, чтобы просмотреть все файлы, хранящиеся в кэше, щелкните на кнопке Просмотр файлов (View Files). IE для Macintosh не предоставляет возможности прямого доступа к кэшу. В Netscape 6 и старше можно ввести в строку адреса такой URL: about:cache. В окне браузера появится список содержимого кэша Выбирая ссылки, можно просмотреть файлы, хранящиеся в памяти или на диске. В Navigator 4.x имеется отдельный кэш для изображений, которому соответствует URL about:image-cache.

Смотрите также

Интерактивные (меняющиеся при наведении мыши) изображения описываются в рецепте 12.2. Примеры использования предварительной загрузки изображений можно увидеть в рецептах 10.9, 10.11, 10.12 и 15.8.

12.2. Интерактивные изображения

NN4 IE4

Задача

Необходимо, чтобы изображение, отображающееся в элементе img или элементе input типа image, менялось при наведении на него указателя мыши.

Решение

Решение состоит из двух частей. Первая часть — это многоцелевая, обратно-совместимая функция для смены изображений, называемая setImage(). Данная функция может быть применена к любому количеству изображений на странице. Показанная ниже функция полагается на существование объектов, ссылающихся на предварительно загруженные изображения, показанные в рецепте 12.1:

```
function setImage(imgName, type) {
    if (document.images) {
        if (type — "hilite") {
            document.images[imgName].src - imagesHilite[imgName].src:
            return true:
        } else if (type == "normal") {
            document.images[imgName] src - imagesNormal[imgName].src:
            return true:
        }
    }
    return false;
}
```

Поскольку ранние версии браузеров не поддерживали обработку событий мыши для элементов **img**, изображения на странице следует поместить в элемент, реагирующий на них:

```
<a href="products html' onmouseover="return setImage('products'. 'hilite')"
    onmouseout="return setImage('products'. 'normal')"><img name="products"
    height="20" width="50" border="0" src="img/prodNormal.jpg"
    alt="Продукты"></a>
```

В IE 4, NN 6 и более поздних браузерах можно пропустить элемент a и использовать для запуска функций события элемента **img**. Помимо этого нужно запрограммировать переход на соответствующий адрес при щелчке на изображении:

```
<img name="products" height="20" width="50" src="img/prodNormal.jpg"
onmouseover="return setImage(this.name. 'hilite')"
    onmouseout="return setImage(this.name. 'normal')"
    onclick="location.href='products.html'" alt="Продукты" >
```

Элементы input типа image поддерживают обработку событий мыши в тех же поколениях браузеров, поэтому вы можете использовать уже показанный код, помещая обработчики событий в тег <input>.

Обсуждение

Несмотря на очевидную сложность решения, когда изображение помещается в гиперссылку только для того, чтобы заставить работать смену изображений, оно автоматически наследует поведение гиперссылки. Например, если пользователь проводит мышью над изображением, в строке состояния появляется адрес перехода (если не изменить это поведение с помощью сценария), это удобно для опытных пользователей. Но для общедоступного сайта более важно то, что поисковые машины используют для навигации по сайту значения атрибутов href. Если же вся навигация будет целиком основана на сценариях, поисковая машина не сможет перейти на другие страницы сайта, тем самым уменьшая вероятность того, что пользователь найдет вашу страницу.

Обратите внимание на то, что в обоих фрагментах HTML, показанных в решении, элемент img применяет атрибут name. Хотя этот атрибут имеется во всех вариациях HTML 4.01, стандарту XHTML 1.0 он не соответствует. Если это для вас важно, вместо него следует взять атрибут id, а в обработчике события первым параметром функции **setImage()** должен быть this.id. Кроме того, внутри функции **setImage()** нужно заменить ссылки document.images[imgName] на вызов document.getElementById():

```
document.getElementById(imgName).src = imagesHilite[imgName].src:
```

Если разрабатывать систему интерактивных изображений, работающую только в современных браузерах, назначение обработчиков событий необходимо переместить из кода элементов (тем самым отделяя содержимое от программирования) и использовать распространение событий, а также современные технологии их обработки. Назначение обработчиков событий верхнего уровня (то есть

событий узла document) и сама обработка событий должны происходить во внешнем js-файле:

```
document.onmouseover=setImage;
document.onmouseout=setImage;
```

Тогда тег <img> (который будет помещен внутрь тега a) должен иметь только обычные атрибуты (включая имя класса, идентифицирующее этот элемент как интерактивное изображение). Вот пример XHTML-кода:

```
<img id="products" slacc="swappable" height="20" width="50"
src="img/prodNormal.jpg" alt="Продукты" />
```

Все сложности перенесены в функцию обработчика событий, которая обрабатывает события мыши только для тех элементов, которые относятся к классу swappable:

```
// Функция для смены изображений
function setImage(evt) {
    if (document.images) {
       // приведение к одному виду объектов событий W3C и IE
        evt = (evt) ? evt • ((window.event) ? window.event : null);
        if (evt) {
           // выравниваем свойства W3C и IE
            var elem - (evt.target) ? evt.target :
                       ((evt.srcElement) ? evt.srcElement : null):
           // отбрасываем устаревшие браузеры (elem==null) и статичные
           // элементы
            if (elem && elem.className — "swappable") {
              // состоянием будет управлять тип события
                switch (evt.type) {
                    case "mouseover":
                        elem.src = imagesHilite[elem.id].src:
                        break:
                    case "mouseout":
                        elem.src - imagesNormal[elem.id].src;
                        break:
                }
            }
        }
    }
}
// назначение обработчиков событий верхнего уровня
document.onmouseover=setImage;
document.onmouseout=setImage;
```

В новейших браузерах (IE 5 и старше, а также NN 6 и старше, но не Opera вплоть до версии 6) можно реализовать интерактивные изображения без использования тега img или сценариев. Вместо этого следует задать два правила таблицы стилей для элемента в нормальном и выделенном состоянии, которое можно указать с помощью слова hover. Таблица стилей должна выглядеть так:

```
a#products {background:url(prodNurmal.jpg): display:block: height:20px:
        width:50px}
a#products:hover {background:url(prodHilite.jpg)}
```

В HTML-коде нужно задать элемент, идентификатор которого ассоциирован с правилом:

```
<a id="products" href="products.html">...<a>
```

Для каждого интерактивного изображения необходима отдельная пара CSS-правил, ассоциированных с уникальными идентификаторами. Ключевым моментом, приводящим это правило в действие, является CSS-свойство display. Заданное значение этого свойства заставляет пустую ссылку открыться в блоке указанных в правиле размеров.

Смотрите также

Предварительная загрузка изображений демонстрируется в рецепте 12.1. Работа с объектами описывается в рецепте 3.8. Совместимая обработка событий в разных браузерах рассматривается в рецепте 9.1.

12.3. Смена стиля текста

NN6 IE4

Задача

Следует изменить некоторые свойства стиля текста, уже отображающегося на странице.

Решение

Можно изменять свойства стиля элемента, содержащего текст, как это делается в показанных ниже примерах:

```
elementReference.style.color = "00ff00":
elementReference.style.font = "bolder small-caps 16px 'Andale Mono', Arial, sans-serif":
elementReference.style.fontFamily = "'Century Schoolbook', Times, serif":
elementReference.style.fontSize = "22px":
elementReference.style.fontStretch = "narrower":
elementReference.style.fontStyle = "italic":
elementReference.style.fontVariant = "small-caps":
elementReference.style.fontWeight = "bolder":
elementReference.style.textDecoration = "line-through":
elementReference.style.textTransform = "uppercase":
```

Обсуждение

Многие из свойств CSS влияют на внешний вид текста, расположенного на странице. В IE 6 и NN 6 реализованы большинство свойств CSS2, а также имеются отдельные собственные добавления. Поскольку свойства стиля элемента управляются ассоциированным с элементом объектом style, значения этих свойств можно изменить после того, как страница будет загружена (в IE 4 и NN 6, а также в более новых версиях).

Прежде чем изменять оформление текста, текст должен быть расположен в отдельном элементе, даже если он является частью большего текста в другом элементе. В рецепте 15.2 показан пример того, как преобразовать выделенный пользователем текст в отдельный элемент, пригодный для изменения его стиля.

Нужно учитывать, что имена CSS-свойств, содержащих дефисы, должны быть преобразованы, чтобы соответствовать правилам языка JavaScript. Таким образом, CSS-свойство font-weight должно записываться как fontWeight. Значения, которые должны присваиваться этим свойствам, всегда представляют собой строки, а константы идентичны тем, что используются в CSS. Значения, обозначающие длину, ширину и т. д., должны включать единицу измерения (например, 22px). В табл. 12.1 причислены свойства и указан тип значения.

Таблица 12.1. Свойства, используемые для оформления текста и их возможные значения

Свойство	Описание
color	Цвет текста, указанный в виде шестнадцатеричной тройки (то есть #ff00ff), CSS RGB-значения (то есть rgb(255,0,255) или rgb(100%,0%,100%)) или цветовая константа (то есть green)
font	Комбинированное свойство, составленное из одного или нескольких значений fontFamily, fontSize, lineHeight (этому значению должен предшествовать символ /), fontStyle, fontVariant и fontWeight; или одна из констант: caption, icon, menu, message-box, small-caption или status-bar
fontFamily	Разделенный запятыми список семейств шрифтов в порядке уменьшения приоритета; имена семейств, состоящие из нескольких слов, должны быть заключены в кавычки
fontSize	Значение, характеризующее высоту символов (фиксированную или процентную), относительный размер (larger или smaller), или одна из констант: xx-small, x-small, small, medium, large, x-large или xx-large
fontStretch	Межсимвольное расстояние, задаваемое константой: normal, wider, narrower, ultra-condensed, extra-condensed, condened, semi-condensed, semi-expanded, expanded, extra-expanded, ultra-expanded или none
fontStyle	Наклон символов, задаваемый константой: normal, italic или oblique
fontVariant	Версия шрифта с маленькими заглавными буквами: normal или small-caps
fontWeight	Толщина символов: bold, bolder, lighter, normal, 100, 200, 300, 400, 500, 600, 700, 800 или 900
textDecoration	Дополнительное оформление текста: blink, line-through, none, overline или underline
textTransform	Преобразование регистра символов: capitalize, lowercase, none или uppercase

Дополнительные настройки стиля могут полностью менять вид текстового элемента. Цвет фона элемента (свойство backgroundColor) оказывает значительное влияние на внешний вид текста и на его читаемость. Другие связанные с оформлением текста свойства, такие как textAlign или textIndent, применяются к содержащему текст блочному элементу.

Если вы собираетесь анимировать переходы между различными состояниями, то, чтобы сделать анимацию видимой, следует использовать функции setTimeout() или setInterval(). Просто последовательно присваивать стилю элемента разные значения нельзя, так как современным браузерам свойственно устанавливать задержку перед обновлением содержимого, пока не завершится выполнение текущего потока. Такой подход позволяет ускорить изменение большого числа разных свойств стиля, применяя их все сразу, вместо того чтобы отдельно отображать изменение каждого свойства. Поэтому если вы хотите, например, на некоторое время изменить цвет фона элемента, чтобы привлечь к нему внимание пользователя, необходимо создать функцию, последовательно вызывающую саму себя с помощью setTimeout(). Каждый раз, когда функция запускается, она должна менять цвет заднего фона элемента, идентификатор которого был заранее передан в нее в качестве параметра:

```
function flashBkgnd(elem, count) {
    // если счетчик содержит null, инициализируем его нулем
    count = (count) ? count : 0;
    // получаем значение для последовательных сравнений.
    var currColor = document.getElementById(elem).style.backgroundColor;
    if (currColor == "rgb(255,255,0)" || currColor == "#ffff00") {
        document.getElementById(elem).style.backgroundColor = "#ff0000";
    } else {
        document.getElementById(elem).style.backgroundColor = "#ffff00";
    }
    if (count < 10) {
        // вызываем эту же функцию через 1/10 секунды, увеличивая значение
        // счетчика
        setTimeout("flashBkgnd('" + elem+"'," + ++count + ")", 100);
    } else {
        // предполагаем, что задний фон имеет белый цвет
        document.getElementById(elem).style.backgroundColor = "#ffffff";
    }
}
```

Эта функция использует внутренний счетчик, значение которого используется как второй параметр при последующих вызовах. После того как счетчик достигнет своего конечного значения, цвет фона элемента должен вернуться к исходному значению. Чтобы определить эффективное значение цвета заднего фона элемента, можно взять рецепт 11.12 и, используя эти данные, восстанавливать исходный цвет элемента.

Кроме того, обратите внимание на то, что в функции flash Bkgrnd() текущее значение цвета проверяется в двух форматах: значение CSS rgb(x,y,z) и значение в виде тройки шестнадцатеричных чисел. Это делается потому, что некоторые браузеры всегда возвращают значение цвета в формате RGB, невзирая на то, как это значение было присвоено.

Смотрите также

Использование специального CSS-класса hover для управления видом гиперссылки описывается в рецепте 12.2. Применение setTimeoutO как средства задержки выполнения описано в рецепте 4.5. Как определить эффективное значение стиля

для элемента, рассказано в рецепте 11.12. Преобразование выделенного пользователем текста в пригодный для оформления с помощью стилей элемент демонстрируется в рецепте 15.2.

12.4. Как выбрать размер шрифта

NN6 IE5

Задача

Необходимо предоставить посетителям возможность выбирать относительный размер шрифта для текста на странице.

Решение

Создайте элемент интерфейса, позволяющий выбирать один из трех размеров шрифта. Каждый выбор должен вызывать показанную ниже функцию changeStyle(). Эта функция активизирует таблицу стилей, идентификатор которой передан в качестве аргумента, а все остальные таблицы стилей отключает. Все эти таблицы стилей относятся к элементу body. В качестве дополнительной возможности эта функция использует библиотеку cookies.js (см. рецепт 1.9), чтобы сохранять пользовательские настройки между посещениями.

Обсуждение

Это решение состоит из трех частей: HTML-код, таблицы стилей и сценарии. Результатом является небольшой элемент управления на странице, позволяющий пользователю выбрать один из трех размеров шрифта, в соответствии с которым отображается остальной текст на странице. На рис. 12.1 показано, как это должно выглядеть.

HTML-код для этого элемента определяет один элемент **div** и несколько вложенных в него элементов **img**. С тремя из этих изображений связаны гиперссылки, вызывающие функцию changeSizetyleO, служащую для управления видом страницы. Чтобы изображения всегда располагались вплотную друг к другу, следует избегать переносов строк между ними в HTML-коде. В показанном ниже примере разрывы строк происходят внутри тегов:

```
<div id="textSizer">
<img src="fontSizer.jpg" height="18" width="72" alt="Выбор размера"><a
href="" onclick="changeSizeStyle('smallStyle'); return false"><img
class="textSize" src="fontSmall.jpg" height="18" width="18" alt="Маленький" /></a><a href=""
onclick="changeSizeStyle(''); return false"><img
class="textSize' src="fontMedium.jpg"  height="18" width="18"
alt="Обычный" /></a><a
href="" onclick="changeSizeStyle('largeStyle'); return false"><img
class="textSize" src="fontLarge jpg" height="18" width="18" alt="Большой" /></a>
</div>
```

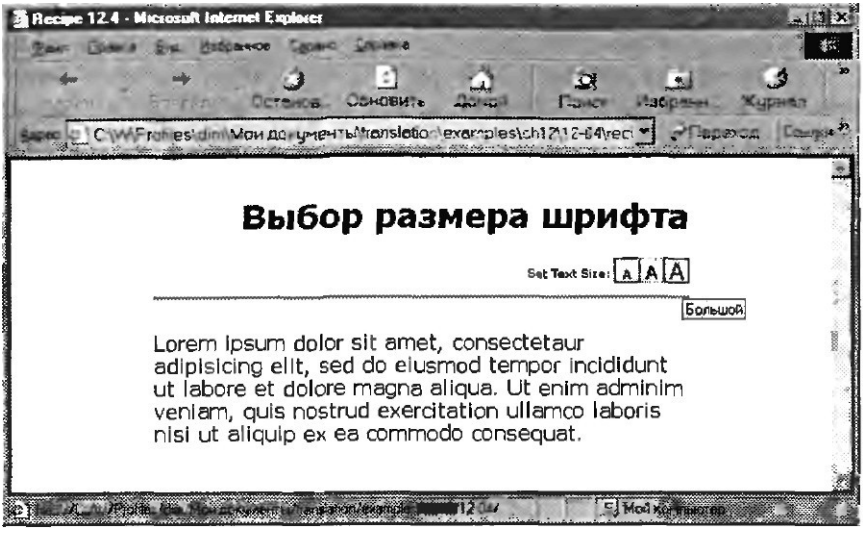


Рис. 12.1. Элемент управления размером шрифта

Другая часть этого рецепта состоит из трех отдельных элементов style (стили также можно импортировать с помощью элемента link). Каждому элементу style назначен идентификатор (атрибут id), используемый при выполнении сценария. По умолчанию элемент управления шрифтом не отображается, как показано ниже, о его отображении заботится сценарий:

```
<style id="normalStyle" type="text/css">
body {font-family:Verdana, Helvetica, sans-serif;
      font-size:small}
#textSizer {text-align:right; display:none}
.textSize {border:1px solid black}
</style>
<style id="sizer" type="text/css" disabled="disabled">
    @import url("textSizer.css");
</style>
<style id="smallStyle" type="text/css" disabled="disabled">
    @import url("smallFont.css");
</style>
<style id="largeStyle" type="text/css" disabled="disabled">
    @import url("largeFont.css");
</style>
```

Каждая из импортируемых таблиц стилей состоит из одного правила Одна из таблиц, textSizer.css, определяет внешний вид регулятора:

```
#textSizer {display:block}
```

Таблица **smallFont.css** содержит такое правило:

```
body {font-size:xx-small}
```

В largeFont.css указано следующее:

```
body {font-size:large}
```

Сценарий в этом решении использует библиотеку cookies.js и содержит две функции, а также оператор, вызывающий одну из них при загрузке страницы:

```
<script type="text/javascript">
// включаем/отключаем
function setSizeStyle() {
    if (document.getElementById) {
        document.getElementById("sizer").disabled = false;
        var styleCookie = getCookie("fontSize");
        var styleIDs = ["smallStyle", "largeStyle"];
        for (var i = 0; i < styleIDs.length; i++) {
            if (styleCookie == styleIDs[i]) {
                document.getElementById(styleIDs[i]).disabled = false;
            } else {
                document.getElementById(styleIDs[i]).disabled = true;
            }
        }
    }
}
// устанавливаем активную таблицу стилей перед тем, как отобразится
// содержимое.
setSizeStyle();

// вызывается при щелчках на значках регулятора размера
function changeSizeStyle(styleID) {
    setCookie("fontSize", styleID, getExpDate( 180, 0, 0));
    setSizeStyle0;
}
</script>
```

Сделанный пользователем выбор сохраняется в cookie и применяется при следующем посещении страницы.

Несмотря на относительно небольшое количество CSS, HTML и JavaScript кода, использованного в этом решении, значительная его часть нужна только для того, чтобы, нормально работая в современных браузерах, код корректно уменьшал бы свою функциональность в более старых.

Регулятор состоит только из изображений, чтобы, несмотря на различные настройки размеров, текст подписи не менялся. Хотя можно задать метке атрибут стиля, отменяющий действие настроек, применить изображение проще. Если же позволить подписям менять свой размер, положение управляющих значков будет меняться при каждом изменении, раздражая пользователей. В показанном здесь примере для размещения регулятора используется не позиционируемый элемент, а для его привязки задается размер левого поля в процентах. Вы можете посчитать более удобным взять абсолютно позиционируемый элемент, привязанный к какому-то другому элементу на странице.

Таблицы стилей сделаны в расчете на то, чтобы в старых браузерах, понимающих CSS, но не воспринимающих применяемый для манипуляций со стилями синтаксис W3C DOM, текст имел бы обычный размер. Также обратите внимание на то, что в этом рецепте не задается никаких фиксированных размеров. Задание размера шрифта всегда было довольно сложной задачей из-за влияния настроек браузера и операционной системы. Используя относительные размеры, мы тем самым позволяем посетителям самостоятельно задавать основной раз-

мер шрифта, предоставляя им на выбор более мелкие или более крупные буквы. Это также означает, что и другие более детальные установки размеров шрифтов на странице относительные, а не абсолютные. Так можно задать шрифт заголовка в em, получив нужный увеличенный размер, но этот размер будет исчисляться относительно размера шрифта во внешнем элементе. В некотором смысле такой подход учитывает недостатки браузеров, предоставляя точное определение вида пользователю. Задача дизайнера в этом случае — убедиться, что содержимое нормально выглядит при разных настройках.

Альтернативный внешний вид обеспечивают три элемента style, изначально отключенные. Для задания атрибута disabled в них используется обратно-совместимый, но в то же время дружественный синтаксис XHTML. Например, хотя заданные по умолчанию стили полностью прячут регулятор с экрана, одна из трех отключенных таблиц стилей способна показать его пользователю (когда эта таблица включена). Это верно и для других шрифтовых таблиц стилей в теле документа. Так как три других элемента style расположены после того, как заданы стили по умолчанию, после включения одного из элементов его свойства получают приоритет (согласно правилам CSS). Таким образом, задача сценария — включать и выключать нужные стили, когда это необходимо. Внешние таблицы стилей здесь необходимы для того, чтобы Netscape 4 не применял их, несмотря на установленный атрибут disabled.

Чтобы выбранные пользователем настройки сохранились до следующего посещения, сценарий рассчитывает на процедуры записи и чтения cookie из библиотеки cookies.js, показанной в рецепте 1.9. Поэтому убедитесь, что выше по коду эта библиотека загружена.

Большая часть работы выполняет функция setSizeStyle(). Она проверяет наличие метода getElementById(), и если метод поддерживается, сначала включает таблицу стилей, делающую видимой регулятор (таким образом, регулятор появляется только в IE 5 и NN 6, а также в более новых браузерах). Затем проверяется, не остался ли с прошлого посещения cookie. Если cookie найден, хранящееся в нем значение становится идентификатором активного стиля, в то время как остальные стили отключаются. Отключать использующийся по умолчанию стиль не нужно, так как новые стили меняют свойство font-size, а все остальные настройки оставляет неизменными. Показанный в этом примере механизм позволяет реализовать дополнительные настройки размеров, если вы захотите предложить больше трех вариантов. Необходимо просто добавить новый элемент style, назначив ему уникальный идентификатор, который также следует вписать в массив внутри функции, а также во второй параметр обработчика события onclick для значка, соответствующего новой настройке.

Так как единственными объектами, с которыми работает функция, являются элементы style, объявленные выше по коду страницы, функция может начать работу до того, как страница загрузится целиком. В показанном примере вызов функции помещен сразу после описания, вызывая ее после загрузки головной секции документа.

Когда пользователь щелкает на одном из значков регулятора размера, обработчик события onclick вызывает функцию changeSizeStyleO, во втором параметре передавая ей идентификатор таблицы стилей, которую нужно активизировать. Эта функция жизненно необходима для работы примера.

Смотрите также

Утилиты для работы с cookie смотрите в рецепте 1.9. Как импортировать таблицы стилей, рассказывается в рецепте 11.4.

12.5. Создание стилей ссылок

NN6 IE4

Задача

Необходимо сделать так, чтобы гиперссылки на странице реагировали на щелчки и перемещение мыши иначе, чем это делается при обычных настройках.

Решение

В CSS существуют псевдоклассы, позволяющие задать разные стили для разных состояний ссылки. В показанном ниже примере текст ссылки имеет темный красновато-коричневый цвет, пока пользователь не выберет ссылку, после чего цвета текста и фона инвертируются. Кроме того, гиперссылка подчеркивается не всегда, как это делается обычно, а только тогда, когда над ней находится указатель мыши.

```
<style type="text/css">
a:link {color:#993300: background-color.#ffffff: text-decoration:none}
a:visited {color:#993300; background-color:#ffffff; text-decoration:none}
a:active {color:#ffffff: background-color:#993300; text-decoration:none}
a:hover {text-decoration:underline}
</style>
```

Обсуждение

Псевдоклассы CSS a:link, a:visited и a:active являются современными версиями использовавшихся ранее атрибутов тега <body>. Эти устаревшие атрибуты контролировали только цвет гиперссылки в одном из трех состояний. Но CSS-версия дает гораздо больший контроль над внешним видом текста и другими окружающими факторами.

Еще одно состояние, которому соответствует псевдокласс a:hover, дает простой способ запрограммировать смену стиля во время того, что программист назвал бы событием mouseover. Этот псевдокласс позволяет сделать интерактивные изображения, не применяя сценарии (см. рецепт 12.2). Все еще необходимо использовать обработку события mouseover потому, что этот способ поддерживается более старыми браузерами, чем те, которые поддерживают псевдоклассы CSS (IE 4 и старше, Opera 5 и старше и NN 6 и старше).

При использовании псевдокласса aiactive в IE для Windows есть одна особенность. Этот браузер по-своему понимает активную ссылку. В то время как

остальные браузеры читают, что ссылка находится в этом состоянии только во время нажатия кнопки мыши на элементе, IE трактует это состояние как наличие фокуса на элементе. То есть если пользователь щелкнет мышью на ссылке, ее видом будет управлять правило для a:active, пока ссылка не потеряет фокус. Кроме того, пользователь может установить фокус на ссылку иначе, нажав достаточное число раз клавишу Tab.

Обдумывая возможность убрать традиционное подчеркивание с гиперссылки, разумно задаться вопросом, нужно ли это делать. Некоторые дизайнеры возражают против стандартного оформления, основываясь на соображениях эстетичности, так как жирный шрифт и подчеркивание отвлекают пользователя от основного содержимого и могут подтолкнуть пользователя перейти на другой сайт слишком рано. Вместо того чтобы избегать использования ссылок (что является признаком не очень-то хорошего сайта), дизайнер может их замаскировать.

Скорее всего, на ваше решение относительно ссылок должна более всего влиять целевая аудитория. Неопытные посетители общедоступного сайта могут просто не заметить блекло окрашенные, неподчеркнутые фрагменты текста, являющиеся ссылками, и тем самым пройти мимо основных ценностей сайта. С другой стороны, опытные пользователи могут инстинктивно догадываться, где должны быть расположены ссылки, и легко заметят ссылки в любой цветовой схеме. Какой бы подход вы ни применяли, важным шагом к тому, чтобы пользователи приняли ваш дизайн, является единый и логичный стиль всего сайта.

Смотрите также

Замена изображений по команде события mouseover демонстрируется в рецепте 12.2.

12.6. Фоновые цвета и изображения

NN6 IE4

Задача

Необходимо дать посетителям возможность выбирать фоновый цвет или фоновое изображение для сайта.

Решение

Следует изменить значения одного или нескольких свойств элемента body, задающих фон. Ниже показан пример, демонстрирующий диапазон возможностей:

```
document.body.background = "url(watermark.jpg) repeat fixed";
document.body.backgroundAttachment = "fixed";
document.body.backgroundColor = "rgb(255. 255. 204)";
document.body.backgroundImage = "url(corp%20watermark.jpg)";
document.body.backgroundPosition = "5% 5%";
document.body.backgroundRepeat = "repeat-y";
```

Обсуждение

Ряд CSS-свойств контролируют внешний вид фона элементов. Если применить эти свойства к элементу body, меняется фон всей страницы. Все доступные для программ свойства и диапазон пригодных для них значений перечислены в табл. 12.2.

Таблица 12.2. Свойства фона и их значения

Свойство	Описание
background	Комбинированное свойство, состоящее из разделенных запятыми значений backgroundAttachment, backgroundColor, backgroundImage, backgroundPosition и backgroundRepeat
backgroundAttachment	Описывает, как фоновое изображение привязано к странице. Возможные значения: fixed и scroll
backgroundColor	Цвет, заданный тройкой шестнадцатеричных значений (то есть #ff00ff), CSS RGB (то есть rgb(255,0,255) или (100%,0%,100%)) или константой (например, red)
backgroundImage	URL фонового изображения в формате CSS (например, url(logo.jpg))
backgroundPosition	Смещение фонового изображения относительно границ элемента. Значение представляет собой пару разделенных пробелом координат или комбинацию из пары констант: bottom, center, left, right или top. Одиночное значение применяется к обоим измерениям
backgroundRepeat	Контролирует, должно ли фоновое изображение повторяться, а также должно ли оно повторяться вдоль отдельной оси. Соответствующие константы: no-repeat, repeat, repeat-x и repeat-y

Если задать одновременно фоновый цвет и фоновое изображение, изображение будет наложено на цветной фон, который просвечивает во всех прозрачных участках

Предоставление пользователям возможности выбирать стиль оформления фона для страниц (например, с помощью элемента select где-либо на странице) — это еще один шаг на пути к дружественному интерфейсу. Выбранное значение нужно сохранять, чтобы при последующем посещении можно было восстановить настройки. Эту информацию можно сохранить в любой базе данных, но гораздо удобнее использовать cookie, хранящиеся на стороне клиента. Это позволяет пользователю избежать необходимости регистрироваться, а затем проходить процедуру авторизации, чтобы просто восстановить предыдущие настройки.

В последующем описании предполагается, что загружена многоцелевая библиотека для записи и чтения cookie из рецепта 1.9 (библиотека предоставляет функции setCookie() и getCookie()). В примере демонстрируется возможность выбора одного из фоновых изображений, список которых размещен в элементе select с идентификатором bgChooser. Последний сделанный пользователем выбор сохраняется в cookie и восстанавливается при последующей загрузке страницы (для этого используется обработчик события onload тега <body>). Стержнем решения является функция, считывающая значение cookie и помещающая его

в свойство backgroundImage. Если cookie пуст, по умолчанию используется первое значение из списка в элементе select:

```
// сохраняем сделанный пользователем выбор в cookie и задаем стиль
function savebgImage(evt) {
    evt = (evt) ? evt   ((event)  ? event : null);
    if (evt) {
        var elem = (evt.target) ? evt.target : evt.srcElement;
        setCookie("bgImage". elem value. getExpDate(180. 0. 0);
       // вызов функции. меняющей видимое изображение
        setbgImage();
    }
}
// меняем фоновое изображение после выбора нового или после загрузки
// страницы
function setbglmage0 {
    var uri = getCookie("bgImage");
    // получаем ссылку на элемент select
    var selector = document.getElementById("bgChooser");
    if (uri) {
       // задаем фоновое изображение на основе значения cookie
        document.body.style.backgroundImage = "url(" + uri + " ) " :
       // в onload устанавливаем select в соответствии со значением cookie
        for (var i = 0: i < selector.options.length: i++) {
            if (uri == selector.options[i].value) {
                selector.options[i].selected = true:
                break:
            }
        }
    } else {
       // если cookie нет. устанавливаем значение. выбираемое по умолчанию
        document.body.style.backgroundImage = "url(" + selector.value + " ) " :
    }
}


<body onload="setbgImage()">

<select id="bgChooser" onchange="savebgImage(event)">
    <option value="deskl.gif">Desk l</option>
    <option value="desk2.gif">Desk 2</option>
    <option value="desk3.gif">Desk 3</option>
    <option value="desk4.gif">Desk 4</option>
</select>
```

Обратите внимание на одну тонкую, но важную деталь функции setbgImage(). Пункт, выбранный в элементе select, всегда должен соответствовать значению cookie. Для этого служит цикл, перебирающий все варианты выбора в select. Опция, значение которой совпадает с cookie, выделяется.

Обсуждение

В рецепте 12.4 демонстрируются альтернативные способы сохранения и восстановления настроек оформления, сделанных с помощью CSS. Библиотека для работы с cookie демонстрируется в рецепте 1.9.

12.7. Управление видимостью элементов

NN6 **IE5**

Задача

Скрыть видимый элемент или показать невидимый элемент.

Решение

Видимость элементов определяют два свойства CSS (и два соответствующих им свойства объектов), но то, какое из свойств вы изберете для управления, значительно повлияет на результаты скрытия и отображения элемента. Наименее радикальное — это style.visibility, поддерживающее значения **hidden, inherit** и **visible.** Ниже показан пример, скрывающий элемент с экрана:

```
document getElementById("warning").style.visibility = "hidden":
```

Изменяя это значение, вы не влияете на остальную страницу. Другое свойство, **style.display,** полностью исключает элемент из процесса отображения страницы, заполняя оставшееся пустое пространство другими элементами, если поместить в него значение none:

```
document.getElementById("warning").style.visibility = "none":
```

Если потом поместить в это свойство подходящее значение (например, **inline** или **block),** элемент будет вновь вставлен в страницу, а прочие ее части передвинутся, освобождая ему место.

Обсуждение

От механизма наследования в CSS зависит результат скрытия или отображения. По умолчанию свойство **visibility** имеет значение **inherit,** то есть видимостью управляет родительский элемент. Если скрыть контейнер, его содержимое тоже скроется. Но в большинстве браузеров элемент сам может контролировать свою видимость. Например, если в Netscape 6 присвоить свойству **visibility** контейнера значение **hidden,** а свойству вложенного в него элемента — **visible,** последний будет виден, несмотря на то, что контейнер видим не будет.

В общем и целом, настройка **visibility** наиболее надежно работает как в блочных, так и в абсолютно позиционируемых элементах. Фактически она поддерживается даже тегами div и **span** в Netscape 4 (а также нестандартными объектами **layer).**

С точки зрения CSS, настройка значения **display** — более серьезное действие. Это свойство может принимать множество разных значений, отвечающих тому, как браузер отображает таблицы, компоненты таблиц и пункты списков. С его помощью можно даже так изменить характеристики, что текстовый элемент станет блочным. Таким образом, помещая в него значение none, вы не просто

скрываете его с экрана: вы тем самым указываете браузеру уменьшить пространство, занимаемое элементом, до нуля и подстроить его окружение. Тем не менее такой элемент не исключается из дерева узлов документа.

Смотрите также

В нескольких рецептах из главы 10 используется скрытие и отображение блоков для формирования динамических меню.

12.8. Настройка прозрачности

NN6 IE4(Win)

Задача

Сделать элемент частично прозрачным, чтобы сквозь него просвечивал фон.

Решение

В CSS Level 2 не предлагается никаких способов описания прозрачности, но в Internet Explorer 6 для Windows и в Netscape 6 существуют различные способы задать прозрачность при помощи CSS-подобного синтаксиса. Фактически IE предлагает два разных способа, один из которых работает только в IE 5.5 для Windows и старше.

Синтаксис, применимый в IE, начиная с 4-й версии, использует свойство filter. Следующее описание делает элемент с идентификатором watermark прозрачным на 75 %:

```
#watermark {filter:alpha(opacity=25)}
```

Более новый синтаксис использует элемент ActiveX, поставляемый с IE 5.5 для Windows, и требует более явного описания:

```
#watermark {filter:progid:DXImageTransform.Microsoft.Alpha(opacity=25)}
```

В Netscape 6 и более новых версиях следует взять свойство, использующееся в Mozilla для управления прозрачностью, -moz-opacity:

```
#watermark { moz opacity:25%}
```

В названии этого свойства на первом месте намеренно поставлен дефис, это гарантирует, что такой синтаксис не будет конфликтовать с будущими дополнениями, которые может ввести W3C.

Обсуждение

Стили фильтрования в Internet Explorer предоставляют широкий набор преобразований, ориентированных преимущественно на текст. Все фильтры задаются

правилами наподобие правил CSS, с именем свойства filter. В первой, совместимой версии значение свойства состоит из имени фильтра с двумя круглыми скобками. Некоторые фильтры, в том числе и alpha, управляющий прозрачностью, требуют дополнительных параметров, задаваемых с помощью знака равенства внутри скобок. Если указывается несколько параметров, их следует разделять запятыми.

Прозрачность можно задавать не только напрямую, указав ее уровень (число от 0 до 100, где 100 означает полную непрозрачность). Существуют дополнительные стили прозрачности, например градиентная прозрачность, меняющаяся при перемещении по элементу. Возможны следующие настройки градиента: однородный (0, по умолчанию), линейный (ему соответствует 1), круговой (значение 2) и прямоугольный (значение 3). Чтобы указать уровень прозрачности в начале градиента, следует использовать параметр opacity, а прозрачность в конце контролирует параметр finishopacity:

```
{filter:a1 pha(opacity=25. finishopacity=75. style=2)}
```

С помощью других дополнительных параметров можно задать координаты начала и конца градиента (свойства **startX**, **startY**, finishX и finishY).

Если необходимо изменять прозрачность с помощью сценария, к этому свойству можно обратиться точно так же, как и к другим значениям стиля, и присвоить ему строковое значение, содержащее имя фильтра, скобки и дополнительные параметры:

```
document.getElementById("myBox").style.filter = "alpha(opacity=80)";
```

Если вы предпочитаете использовать новый элемент управления ActiveX, имеющийся в IE 5.5 и старше, ссылки на него должны включать внутренний путь к элементу, как это показано в решении. Но хотя ссылки должны содержать эту информацию, доступ к параметрам прозрачности производится через коллекцию filters элемента, а не через объект styles:

```
document.getElementById("myBox").
   filters["DXImageTransform.Microsoft.Alpha"].Opacity=80;
```

Используя такой синтаксис, убедитесь, что именованный фильтр объявлен в CSS или в атрибуте style самого элемента. Элементу, которому не был с помощью стилей назначен фильтр **DXImageTransform**, изменять таким способом прозрачность из сценария нельзя.

Программное управление прозрачностью в Mozilla осуществляется через свойство **style.** MozOpacity. Значением этого свойства может быть либо значение от 0 до 100, либо от 0 до 100% (включая символ процента). Обратите внимание на то, что первая буква имени свойства набрана в верхнем регистре. Прозрачность в Mozilla всегда однородная и не поддерживает градиентов.

Смотрите также

Как полностью скрыть элемент, показано в рецепте 12.7.

12.9. Создание эффектов перехода

NN нет **IE4(Win)**

Задача

Необходимо, чтобы изменения внешнего вида элемента (то есть его отображение, скрытие, смена изображения) происходили с помощью соответствующего визуального эффекта, такого как «стирание», «шторки», «шахматная доска» или «растворение».

Решение

Переходы в IE для Windows являются частью расширений CSS для фильтрования изображений (в IE для Macintosh не реализованы). Возможны два варианта синтаксиса, один из которых совместим с браузерами вплоть до IE 4, другой требует использования IE 5.5 и старше. Здесь демонстрируется эффект растворения при смене изображения в рецепте 12.2.

Описание перехода состоит из двух частей. Первая часть — это определение фильтра, заданное через правила CSS. Чтобы получить эффект перехода с помощью совместимого синтаксиса, тег <img> из рецепта 12.2 необходимо модифицировать следующим образом:

```
<img name="products"  height="20" width="50" border="0"
src="img/prodNormal.jpg" style="filter:blendTrans(duration=0.5)"
alt="Products">
```

Новый вариант, в котором применяется более мощный элемент управления ActiveX, выглядит так:

```
<img name="products"  height="20" width="50" border="0"
src="img/prodNormal.jpg" alt="Products"
style="filter:progid:DXImageTransform.Microsoft.Fade(duration=0.5)" >
```

Вторая часть перехода состоит из двух методов объекта filter: apply() и play(). Метод applyO фиксирует вид элемента, к фильтру которого вы обращаетесь. Это дает возможность изменить настройки, пока элемент вне поля зрения. После этого метод play() осуществляет переход между старым состоянием и только что заданным. При использовании совместимого синтаксиса функцию смены изображения из рецепта 12.2 следует изменить так:

```
function setImage(imgName, type) {
    if (document.images) {
        document.images[imgName].filters["blendTrans"].apply();
        if (type = "hilite") {
            document images[imgName].src = imagesHilite[imgName].src;
        } else if (type == "normal") {
            document.images[imgName].src = imagesNormal[imgName].src;
        }
```

```
        document.images[imgName].filters["blendTrans"].play();
        return true;
    }
    return false;
}
```

Если же использовать для управления фильтрами ActiveX, функция должна выглядеть следующим образом:

```
function setImage(imgName, type) {
  if (document.images) {
    document.images[imgName].
        filters["DXImageTransform.Microsoft.Fade"].apply();
    if (type == "hilite") {
        document.images[imgName].src = imagesHilite[imgName].src;
    } else if (type = "normal") {
        document.images[imgName].src = imagesNormal[imgName].src;
    }
    document.images[imgName].
     filters["DXImageTransform.Microsoft.Fade"].play();
    return true;
  }
  return false;
}
```

Еще одно различие между функцией из рецепта 12.2 и приведенными выше вариантами состоит в том, что оператор return перемещен так, чтобы выполняться после перехода.

Обсуждение

Два разных поколения CSS-фильтров в IE используют для обращения к отдельным эффектам перехода совершенно разные способы. Переходы совместимого типа разбиты на две группы: смешивание (растворение) и появление (разных типов). Смешивание задается с помощью фильтра blendTrans(), один из параметров которого задает длительность перехода в секундах:

```
img.blends {filter:blendTrans(duration=0.5)}
```

Переход типа «появление» (фильтр revealTrans()) характеризуется двумя параметрами: transition, представляющим собой целое число, характеризующее тип перехода, и duration, задающим длительность:

```
div.wipe {filter:revelTrans(transition=7, duration=l 5)}
```

Различные типы переходов перечислены в табл. 12.3.

Свойствами отдельного фильтра можно управлять из сценария. Например, если нужно изменить тип перехода для отдельного элемента с перелистывания на расширяющийся круг, следует применить такой оператор:

```
ссылкаНаЭлемент.filters["revealTrans"].transition = 2;
```

При использовании нового синтаксиса каждому типу перехода соответствует отдельный фильтр, как показано в табл. 12.4.

Таблица 12.3. Типы переходов в IE (совместимый вариант)

Тип	Значение	Тип	Значение
0	Сужающийся прямоугольник	12	Случайное растворение
1	Расширяющийся прямоугольник	13	Вертикальная сходящаяся щель
2	Сужающийся круг	14	Вертикальная расходящаяся щель
3	Расширяющийся круг	15	Горизонтальная сходящаяся щель
4	Перелистывание сверху	16	Горизонтальная расходящаяся щель
5	Перелистывание снизу	17	Левый нижний уголок
6	Перелистывание справа	18	Левый верхний уголок
7	Перелистывание слева	19	Правый нижний уголок
8	Вертикальные шторы	20	Правый верхний уголок
9	Горизонтальные шторы	21	Случайные горизонтальные полосы
10	По клеткам, поперек	22	Случайные вертикальные полосы
11	По клеткам, вдоль	23	Произвольный вариант

Таблица 12.4. Фильтры переходов в IE (новый вариант)

Имя фильтра	Описание
Barn()	Эффект шторок, характеризуемый параметрами duration, motion и orientation
BlindsQ	Эффект жалюзи, свойства: direction, duration и ширина реек (bands)
CheckerboardO	Эффект шахматной доски, свойства direction, duration и размер клеток (squaresX и sqaresY)
Fade()	Постепенный переход, задаваемый длительностью (duration) и степенью перекрытия обеих видов (overalap)
GradientWipe()	Эффект перелистывания с градиентом вдоль линии перехода, свойства: duration, ширина градиента (gradientSize) и направление (wipeStyle)
Insert()	Эффект перелистывания вдоль вертикальной и горизонтальной оси, диагонально из одного угла в противоположный. Характеризуется длительностью (duration)
Iris()	Эффект увеличения. Свойства: duration, направление (motion: in или out) и стиль: irisStyle (circle, cross, diamond, plus, square, star)
Pixelrate()	Переход между видами с помощью расширения и сжатия, а также размытия и фокусировки, характеризуемый параметрами duration (длительность) и максимальным размером блока (maxSquare)
RadialWipe()	Круговой переход, задаваемый длительностью (duration) и стилем wipeStyle (clock, wedge, radial)
RandomBars()	Переход с помощью случайных полосок, характеризуется длительностью (duration) и направлением (orientation)

продолжение

Таблица 12.4 (продолжение)

Имя фильтра	Описание
RandomDissolve()	Переход с помощью случайной замены пикселов, задается длительностью (duration)
Slide()	Скользящие полосы разных типов, задается шириной полос (bands), длительностью (duration) и стилем slideStyle (hide, push, swap)
Spiral()	Переход по спирали, параметры: duration и размер спирали (gridSizeX, gridSizeY)
Stretch()	Переход с помощью растяжения или сжатия, параметры: duration и stretchStyle (значения hide, push, spin)
Strips()	Переход с помощью эффекта полос, характеризуется длительностью (duration) и направлением (motion)
Wheel()	Эффект колеса со спицами, с центром посредине элемента. Параметры: duration и размер спиц (spokeSize)
ZigZag()	Переход с помощью удаления рядов блоков, характеризуется длительностью (duration) и размером блоков (gridSizeX, gridSizeY)

Новый механизм фильтров имеет гораздо больше возможностей, чем обратно-совместимый, к тому же он значительно более многословный, потому что при обращении к фильтру следует обращаться к элементу ActiveX:

```
document.images[imgName].filters["DXImageTransform.Microsoft.Fade"].apply();
```

Если вам необходимо изменить стиль перехода уже после того, как страница загружена, это можно сделать при помощи свойства style.filter:

```
ссылкаНаЭлемент.style.filter =
    "progid: DXImageTransform.Microsoft.Iris(duration=1.0)";
```

Обращаться к стилям фильтрования довольно сложно, поскольку синтаксис меняется в зависимости от ваших намерений. Чтобы управлять существующим фильтром заданного типа (то есть чтобы вызвать один из его методов или изменить свойства), следует задействовать массив filters у самого элемента. Индексом в этом массиве может служить либо число (номер фильтра по порядку в коде страницы), либо строка с именем фильтра. Чтобы изменить тип фильтра, нужно поместить новое описание в его свойство style.filter.

Методы apply() и play() работают в пределах одной страницы, если менять характеристики отображения одного из элементов. Но если необходимо применить эффекты перехода в слайд-шоу, слайды в котором являются отдельными HTML-страницами, описание эффекта следует поместить в теги <meta>. Отдельные значения атрибутов указывают браузеру применять эффект перехода при загрузке и выгрузке страницы. Обычно переход нужен либо только при загрузке, либо при выходе со страницы, но не одновременно. Исключением может быть ситуация, когда у вас есть пустая промежуточная страница между двумя переходами, подчеркивающая оба эффекта, например эффект увеличения при

входе на пустую страницу и уменьшения при выходе с нее. Meta-теги для пустой страницы должны быть заданы так:

```
<meta http-equiv="Page-Enter"
content="progid:DXImageTransform.Microsoft.Iris(Motion='in' IrisStyle='circle')">
<meta http-equiv="Page-Exit"
content="progid:DXImageTransform.Microsoft.Iris(Motion='out' IrisStyle='circle')">
```

После того как вы поместите такое описание на пустую страницу, вам не нужно описывать переходы на страницах с информацией. Также не нужно никаких сценариев. Но вам все равно придется использовать сценарий, чтобы передать в пустую страницу информацию о следующем кадре (например, с помощью аргументов URL, см. рецепт 10.6). Сценарий на пустой странице должен проанализировать аргументы, помещаемые в location.search, и перейти на указанную страницу, тем самым приводя в действие эффект перехода.

Смотрите также

Передача данных между страницами с использованием URL демонстрируется в рецепте 10.6. DHTML слайд-шоу показано в рецепте 15.4. Более подробную информацию о переходах и их свойствах можно найти по адресу **http://msdn.microsoft.com/workshop/author/filter/filters.asp.**

13 Позиционирование элементов HTML

13.0. Вступление

После того как типичная HTML-страница загружена, браузер располагает ее элементы в соответствии со своим пониманием того, как каждый из них должен располагаться на странице. У некоторых элементов есть атрибуты, задающие различные размеры в терминах пикселов или процентов оставшегося свободного пространства, но, опять же, разработчику остается уповать на милость браузера. Из-за того, что HTML изначально позиционировался как пассивный носитель, а не как носитель для печати, где дизайнер полностью контролирует каждый миллиметр страницы, в HTML не предусмотрена тонкая настройка.

Точное размещение содержимого часто производится с помощью HTML-таблиц и невидимых изображений в ячейках таблиц. Например, когда нужно в соответствии с ожиданиями дизайнера расположить изображение и подпись к нему. Если вы когда-нибудь пробовали использовать WISYWIG (What You See Is What You Get) программы для веб-дизайна в той же манере, что и такие программы, как PageMaker или QuarkXpress, то знаете, что за красиво оформленной страницей скрывается масса HTML-таблиц. Ручная правка такого замысловатого кода может буквально довести вас до слез.

Частично благодаря рекомендациям W3C концепция таблиц стилей обрела прочный фундамент в подготовке сетевых публикаций, предоставляя необходимый для этого уровень контроля над видом содержимого. Благодаря ей удалось наполнение страниц (в основном, изображения и текст) отделить от оформления, кроме того, сообщество веб-издателей внесло свой вклад в разработку CSS-свойств, применимых для оформления содержимого.

Вскоре после появления первых рекомендация W3C сформировался дополнительный стандарт, покрывающий основные идеи позиционирования содержимого HTML. В CSS Level 2 позиционирование стало частью основных рекомендаций. В своей основе CSS-позиционирование позволяет задать точное местоположение элемента в пространстве документа. Такой элемент находится на своем собственном слое над основным документом, во многом напоминая прозрачный листок, который аниматоры используют, чтобы создать персонаж, движущийся над статичным фоном. До появления CSS-позиционирования

каждый элемент занимал собственную прямоугольную область на странице. Применяя позиционирование, можно добиться того, чтобы элементы перекрывались, следовательно, должна быть возможность управлять порядком наложения. Наконец, благодаря тому, что позиционируемые фрагменты страницы существуют на отдельных слоях, они могут появляться и исчезать, не влияя на остальную страницу.

Добавив к позиционированию возможность использовать сценарии, имеющуюся в современных браузерах, вы откроете для себя путь к безграничным возможностям конструирования пользовательского интерфейса, далеко выходящим за пределы исходных концепций веб-публикаций. Сценарий может заставить объект прыгать с места на место или же плавно скользить по заданному пути (хотя и не с тем уровнем контроля над скоростью, что в специализированных программах для анимации, таких как Macromedia Flash). Возможность взаимодействия с событиями мыши позволяет отслеживать как элемент под курсором мыши, так и положения самого курсора, благодаря чему можно перетаскивать элементы по странице. Другой вариант анимации позволяет формировать автоматически прокручивающиеся блоки текста.

Область перемещения

Решив применить для оформления страницы позиционирование, необходимо определиться с диапазоном значений, определяемых используемой областью позиционирования. Обычно областью позиционирования является все пространство, занимаемое страницей в окне браузера. В этом случае область, занимаемую страницей, следует рассматривать как холст, на котором нужно разместить один или несколько элементов, каждый в отдельном слое. Положение задается двумя координатами, при этом точка с координатами 0,0 расположена в левом верхнем угле страницы (левый верхний угол области содержимого в браузере, если страница может прокручиваться). Оси координат направлены таким образом, что при движении вниз или вправо координаты увеличиваются.

Областью позиционирования может быть и другой перемещаемый элемент. Другими словами, можно двигать один элемент внутри другого. В этом случае перемещение происходит во внутренней системе координат внешнего элемента, начало которой находится в левом верхнем его углу. Если внешний элемент сдвинется, вместе с ним сдвинутся и вложенные в него, сохраняя свои относительные координаты.

При разработке следует учитывать, что позиционируемый блок, хотя и находится над всем остальным содержимым страницы, не ведет себя как окно браузера. Если передвинуть его так, чтобы занимаемая им область вышла за пределы страницы, элемент, вместо того чтобы выйти за пределы окна, будет обрезан по его границе. Если разместить объект за пределами основной страницы, браузер будет вести себя так, как будто размеры страницы изменились, например, появятся полосы прокрутки, если их не было. Тем не менее, если позиционировать один элемент внутри другого, обрезания не произойдет, если явно не задать отсекающий прямоугольник для контейнера.

Типы позиционирования

Есть несколько способов позиционирования на выбор. Полезным типом, не реализованным в Internet Explorer вплоть до версии 6, является фиксированное позиционирование. Этот тип позволяет неподвижно расположить элемент в окне браузера, независимо от прокрутки страницы. Типичное применение такого типа позиционирования — различные плавающие панели и значки страниц (наподобие знаков, которые можно увидеть в углу многих телевизионных каналов). Чаще употребляются два других типа позиционирования: абсолютное и относительное. Важно понимать возможности этих двух вариантов.

Абсолютно позиционируемый элемент исключается из процесса отображения остальной части страницы и не зависит от положения в ее коде. Абсолютно позиционируемый элемент использует для размещения координаты страницы. Рассмотрим этот момент подробнее, это не так просто. HTML-элементы, следующие за описанием такого объекта, размещаются на странице так, как если бы этого объекта не существовало. Фактически, если сделать блок абсолютно позиционируемым, но не указать ему координаты, он будет размещен в левом верхнем углу страницы, поверх того, что окажется в этом месте страницы. Тем не менее иерархия вложения элементов, учитывающая вложение узлов, все равно имеет место.

Относительно позиционируемый элемент — противоположность абсолютно позиционируемому. Принципиальным отличием между этими двумя способами размещения является то, что относительно позиционируемый блок участвует в процессе размещения основного содержимого документа. Система координат такого элемента начинается там, где он бы располагался без позиционирования. Поэтому если сделать его относительно размещаемым и не указать координат (по умолчанию берутся координаты 0,0), страница будет выглядеть так же, как и без позиционирования. Различие возникает, если задать элементу координаты. Он передвинется в соответствии с ними, а остальное содержимое страницы останется на своем месте.

Важным применением относительного позиционирования является задание системы координат для абсолютно размещаемых элементов. В этом случае относительно размещаемый элемент включается в процесс отображения обычным образом, когда на его положение влияют размер окна, настройки стилей и пр. Вложенный в него блок использует абсолютный тип размещения, благодаря чему можно расположить его относительно страницы, например на 20 пикселов ниже абзаца (см. рецепт 13.2).

Режимы работы

В первых версиях Internet Explorer для Windows, позволявших позиционировать элементы, это делалось способом, совершенно отличным от предложенного позднее W3C в рекомендациях по CSS. Чтобы вернуться на путь стандартов, IE 6 реализует поддержку CSS в большем соответствии со стандартами. Но это может привести к тому, что старые страницы, нормально смотревшиеся в IE от версии 4 до 5.5, отобразятся не так, как ожидается.

Чтобы не допустить этого, IE 6 поддерживает два режима, один — для обратной совместимости и другой — стандартный. По умолчанию IE 6 работает в режиме обратной совместимости. Чтобы перевести его в режим совместимости

с CSS, в начало документа необходимо включить декларацию DOCTYPE с некоторыми параметрами. Netscape 6 и старше также поддерживает две модели, но различия между ними в обычных задачах не столь велики. Информацию о соответствующих декларациях DOCTYPE можно увидеть в рецепте 11.13.

В стандартном режиме работы IE 6 основная система координат для узлов верхнего уровня привязана не к элементу <body> (как это было в IE от 4 до 5.5), а к элементу <html>, означающему весь документ целиком. Это различие далеко не символическое, так как в IE у элемента body имеются встроенные поля, которые могут повлиять на попытки разместить подвижные объекты в соответствии со статичным содержимым страницы (см. рецепт 13.6).

Несовместимости

Хотя сейчас это замечание не столь важно, как раньше, следует учитывать возможные проблемы со страницами, включающими позиционируемые элементы в устаревших или упрощенных браузерах. Если браузер не поддерживает CSS-позиционирование, элемент, который отмечен как подвижный, в действительности будет отображен там, где он описан в коде страницы. Поэтому тщательно размещенные элементы не только не окажутся там, где должны быть, но и могут запутать посетителя.

Борьба с этой проблемой приводит к различным компромиссам, зависящим от особенностей вашего решения. Один из вариантов сводится к тому, чтобы ограничить доступ к странице браузерами, поддерживающими позиционирование (IE 4 и старше, в некоторой степени Navigator 4, Netscape 6 и старше, Opera 5 и старше, а также все остальные браузеры, поддерживающие объект document.body.style).

Другим вариантом является применение сценария для автоматической генерации кода позиционируемых элементов при загрузке страницы. Такой сценарий может определить версию браузера и использовать document.write() для формирования HTML-кода. При этом, конечно, страница должна сохранять свой смысл и в старых браузерах без подвижных элементов оформления. Потенциальным недостатком такой методики является то, что поисковые машины могут не найти ссылки на документы в динамически генерируемом коде.

Единицы измерения

Дизайнеры из издательского мира привыкли выражать размеры в различных единицах, таких как дюймы, сантиметры, цицеро, пункты и шпации. Все эти единицы измерения поддерживаются для задания длин, но большую их часть неудобно использовать, если результат должен выводиться на экран. В мире мониторов король — пиксел. Это, к тому же, неделимая единица, так что любые попытки использовать точность меньше пиксела, обречены на провал.

Но даже если вы рассчитываете выводить результат на принтер, вы можете не получить ожидаемый результат, если используете единицы меры, отличные от пикселов. В браузерах между тем, что видно на экране, и тем, что выводится на принтер, происходит множество преобразований, приближенных вычислений и интерполяций. Да, можно добиться успеха и с шаблоном, использующим другие единицы измерения, но это скорее исключение, чем правило.

Устаревший тег <layer>

Первым браузером, реализовавшем идею позиционирования, был Navigator 4, в котором для этого использовался новый элемент layer. Эта попытка предварила попытку W3C стандартизировать позиционирование в CSS. Когда стандарт был создан, в нем постарались избежать тега <layer> и всего связанного с ним кода. Но Navigator 4 был популярным браузером, и разработчики создали немало страниц, не только применявших тег <layer>, но и требующих поддержки Navigator 4 DOM для использования этих элементов из сценариев. Между тем, Microsoft представил собственную схему обращения к элементам, где каждый из них (позиционируемый и нет) был доступен через коллекцию document.all. Поэтому обход различий между способами доступа к свойствам подвижных объектов в IE и NN 4 в то время был частью любой DHTML- страницы.

После исключения тега <layer> из стандарта HTML разработчики следующих версий Netscape приняли сложное, но совершенно правильное решение выбросить поддержку этого тега и устаревшей схемы доступа из Mozilla (то есть из Netscape 6 и старше). Множество рассчитанных на Netscape сценариев перестали работать после перехода, но благодаря поддержке W3C DOM в современных браузерах сейчас можно использовать один и тот же способ обращения к позиционируемым элементам и манипулирования ими в широком диапазоне разных браузеров, включая IE 5 и старше. W3C DOM — стабильная, развивающаяся платформа для веб-разработчика.

Наследие Netscape 4 сохранилось в том, что позиционируемые элементы часто называют слоями (layers), независимо от браузера. Это удобная метафора того, как ведет себя позиционируемый, подвижный элемент, и вы можете увидеть его во многих местах книги.

13.1. Позиционирование элемента в документе

NN4 **IE4**

Задача

Один из элементов страницы должен перекрывать другие или находиться в определенном месте страницы, независимо от порядка отображения.

Решение

Необходимо присвоить CSS-свойству position значение absolute. Это можно сделать в отдельном описании стиля:

```
<style type="text/css">
#идентификаторЭлемента {position:absolute; left:100px; top:100px}
</style>
```

Или включить описание стиля в тег элемента:

```
<div id="myDIV" style="position:absolute: left:100px: top:100px">
TeKCT</div>
```

Координаты следует задавать относительно левого верхнего угла страницы.

Обсуждение

В таком элементе нет ничего особенно динамического, за исключением возможности последующего изменения его положения, видимости, наложения и отсечения из сценария. За исключением Navigator 4, подвижным может быть любой элемент с любым содержимым, которому также могут быть назначены обработчики событий (например, для его перетаскивания). Даже строчный элемент, такой как span, становится блочным при позиционировании (для этого даже не обязательно менять значение свойства **display).**

Как будет показано в рецепте 13.3, изменение положения объекта на странице производится через свойство элемента style. Например, чтобы изменить вертикальную координату, используя синтаксис W3C DOM, можно применить следующий код:

```
document.getElementById("myDIV").style.top = "200px";
```

Учтите, что свойство **position** доступно только для чтения, то есть если элемент отображен в соответствии с этим значением, оно уже не может быть изменено. Так что нельзя превратить строчный элемент в блочный, просто изменив значение его свойства style.position.

Смотрите также

Библиотека сценариев для управления подвижными элементами приведена в рецепте 13.3. Синтаксис таблиц стилей описывается в главе 11.

13.2. Связывание подвижного элемента с телом документа

NN6 **IE4**

Задача

Подвижный объект должен быть расположен на фиксированном смещении относительно некоторого места в документе.

Решение

Можно создать относительно позиционируемый элемент, привязанный к документу, а в него поместить абсолютно позиционируемый. Следующий пример

кода демонстрирует, как прикрепить изображение копирайта к изображению на странице:

```
<div style='position:relative: margin-left:20%">
    <img src="kitty.jpg" height="511" width="383"
        style="border:4px groove darkred" alt="Мой кот" />
    <img src="photoCopyright.jpg" alt="Copyright 2003 Snaps McGraw"
        style="position:absolute: top:519px: left:263px"/>
</div>
```

Обсуждение

Результат вы можете увидеть на рис. 13.1. Относительно размещаемый контейнер в этом примере использует для задания полей процентную меру, поэтому точное его положение может меняться вместе с шириной окна браузера. Но, где бы ни была расположена фотография, изображение копирайта появится непосредственно под ней, выровненное по правому краю, в следующих браузерах: Internet Explorer 4 и старше (кроме 5.0 для Windows), а также Netscape 6 и старше.

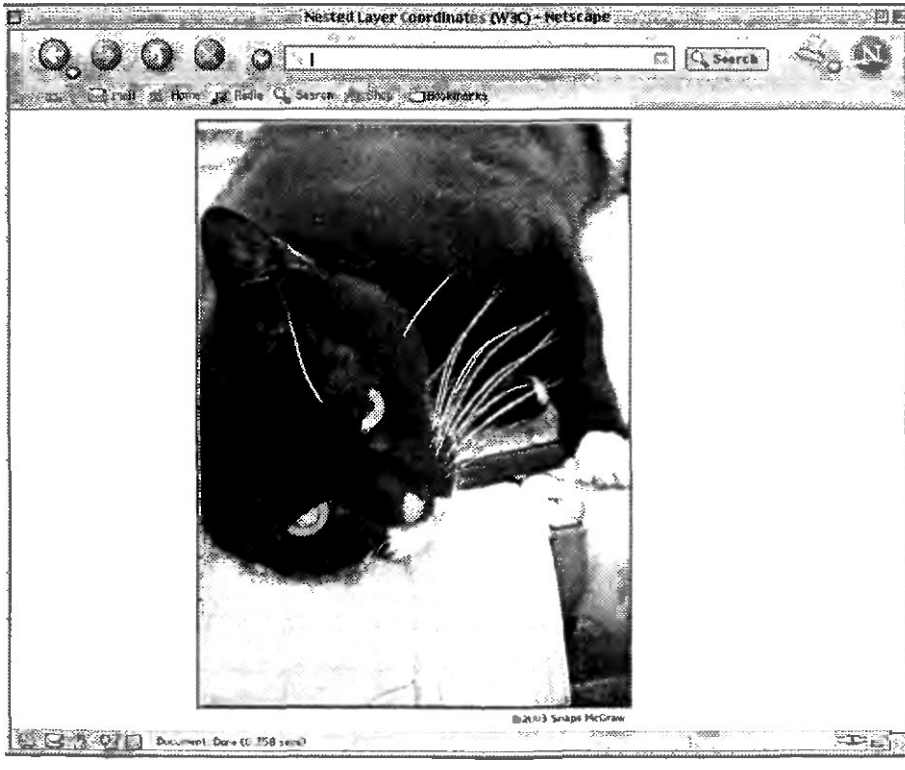


Рис. 13.1. Привязка элемента к относительно позиционируемому объекту

В приведенном решении координаты размещаемого элемента следует подстраивать под каждое изображение в отдельности, так как они зависят от его размеров. Положение точки с координатами 0,0 определяется положением пер-

вого элемента, вложенного в обрамляющий div. Так как основное изображение высотой 511 пикселов имеет рамку шириной 4 пиксела, вертикальная координата изображения с копирайтом должна быть равна высоте изображения вместе с удвоенной шириной рамки, то есть 519 пикселам. Горизонтальная координата равна ширине основного изображения (383 пиксел) минус ширина изображения копирайта (120 пикселов).

Выбор в качестве контейнера элемента div определяется блочной природой изображения в рассматриваемой разметке. Этот тег позволяет легко создать блочный HTML-контейнер там, где это нужно (см. рецепт 13.4). Так как изображение контейнером не является, применять для привязки абсолютно позиционируемого объекта его нельзя, поэтому и нужен контейнер.

Предложенная методика пригодна и для перекрывающихся элементов. Например, вместо изображения копирайта владелец сайта мог бы предпочесть разместить на фоне изображения малозаметную, похожую на водяной знак строчку. Если это изображение имеет те же размеры, что и основное, его можно разместить по координатам 0,0, что соответствует левому верхнему углу основного изображения. Тем не менее, если изображение копирайта имеет меньшие размеры, его можно расположить в любом месте основного, главное, чтобы эта надпись не отвлекала внимания.

Смотрите также

Для понимания различий между разными типами позиционирования смотрите вступление к главе 13. Как использовать для размещения объектов элементы div или span, рассказано в рецепте 13.4.

13.3. Библиотека для управления позиционированием

NN4 **IE4**

Задача

Необходима многоцелевая библиотека, упрощающая управление позиционируемыми элементами на странице из сценария.

Решение

В листинге 13.1 в обсуждении приведена библиотека DHTMLAPI.js, упрощающая работу сценариев с позиционируемыми элементами в разных браузерах. Библиотека совместима с IE 4 и старше, а также с Navigator 4 и старше. Она предоставляет библиотеки для перемещения, скрытия/отображения элементов и для управления порядком их наложения. Например, чтобы передвинуть элемент с идентификатором helpWindow в точку с координатами 300 пикселов

влево и 200 пикселов вниз от левого верхнего угла страницы, можно применить следующий код:

```
shiftTo("helpWindow" 300. 200):
```

Дополнительные вспомогательные функции этой библиотеки помогают получить ссылку на элемент по его идентификатору, его размеры и положение, а также размер окна в некоторых браузерах.

Если библиотека хранится в файле с именем **DHTMLAPI.js**, ее можно подключить к странице следующим тегом:

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

Если странице не назначен обработчик события onload, библиотека инициализируется самостоятельно, в противном случае не забудьте включить в обработчик вызов функции инициализации библиотеки, как это показано в обсуждении.

Обсуждение

Полный код библиотеки DHTMLAPI.js показан в листинге 13.1. В дополнение к набору вспомогательных функции она создает пять глобальных булевских переменных с информацией о браузере, которые также можно использовать в ваших сценариях.

Листинг 13.1. Библиотека DHTMLAPI.js

```
// Глобальные переменные
var isCSS. isW3C, isIE4. isNN4;
// инициализация после загрузки, чтобы все браузеры успели сформировать
// структуру объектов
function initDHTMLAPI() {
    if (document.images) {
        isCSS = (document.body && document.body.style) ? true : false;
        isW3C = (isCSS && document.getElementById) ? true • false:
        isIE4 = (isCSS && document.all) ? true : false:
        isNN4 = (document.layers) ? true : false:
        isIE6CSS = (document.compatMode &&
           document.compatMode.indexOf("CSS1") >= 0) ? true : false:
    }
}
// назначение обработчика события для инициализации API
window.onload = initDHTMLAPI;

// поиск вложенного объекта layer в NN4 по имени
function seekLayer(doc. name) {
    var theObj;
    for (var i = 0: i < doc.layers.length; i++) {
        if (doc.layers[i].name = name) {
            theObj = doc.layers[i];
            break:
        }
```

```
        // переходим к вложенным слоям, если таковые имеются
        if (doc.layers[i].document.layers.length > 0) {
            theObj = seekLayer(document.layers[i].document, name);
        }
    }
    return theObj;
}
// Преобразуем строку с именем объекта или ссылку на объект
// в ссылку на элемент документа
function getRawObject(obj) {
    var theObj;
    if (typeof obj == "string") {
        if (isW3C) {
            theObj = document.getElementById(obj);
        } else if (isIE4) {
            theObj = document.all(obj);
        } else if (isNN4) {
            theObj = seekLayer(document, obj);
        }
    } else {
       // Получаем ссылку на объект
        theObj = obj.
    }
    return theObj;
}

// Преобразуем строку с именем объекта или ссылку на объект
// в ссылку на объект стиля (или в ссылку на слой NN 4)
function getObject(obj) {
    var theObj = getRawObject(obj);
    if (theObj && isCSS) {
        theObj = theObj.style;
    }
    return theObj;
}

// располагаем объект по определенным пиксельным координатам
function shiftTo(obj, x, y) {
    var theObj = getObject(obj);
    if (theObj) {
        if (isCSS) {
           // Преобразуем некорректные числовые значения
            var units = (typeof theObj.left == "string") ? "px" : 0;
            theObj.left = x + units;
            theObj.top = y + units;
        } else if (isNN4) [
            theObj.moveTo(x,y)
        }
    }
}

// Сдвиг объекта на заданный вектор
function shiftBy(obj, deltaX, deltaY) {
    var theObj = getObject(obj);
```

продолжение

Листинг 13.1 (продолжение)

```
    if (theObj) {
        if (isCSS) {
          // преобразуем некорректные числовые значения
           var units = (typeof theObj.left == "string") ? "px" : 0;
           theObj.left = getObjectLeft(obj) + deltaX + units;
           theObj.top = getObjectTop(obj) + deltaY + units;
        } else if (isNN4) {
           theObj.moveBy(deltaX, deltaY);
        }
    }
}

// Устанавливаем порядок наложения объектов (Z order)
function setZIndex(obj, zOrder) {
    var theObj = getObject(obj);
    if (theObj) {
        theObj.zIndex = zOrder;
    }
}

// Задаем фоновый цвет объекта
function setBGColor(obj, color) {
    var theObj = getObject(obj);
    if (theObj) {
        if (isNN4) {
            theObj.bgColor = color;
        } else if (isCSS) {
            theObj.backgroundColor = color;
        }
    }
}

// Делаем объект видимым
function show(obj) {
    var theObj = getObject(obj);
    if (theObj) {
        theObj.visibility = "visible";
    }
}

// Убираем объект с экрана
function hide(obj) {
    var theObj = getObject(obj);
    if (theObj) {
        theObj.visibility = "hidden";
    }
}

// Определение горизонтальной координаты элемента
function getObjectLeft(obj) {
    var elem = getRawObject(obj);
    var result = 0;
    if (document.defaultView) {
        var style = document.defaultView;
        var cssDecl = style.getComputedStyle(elem, "");
```

```
        result = cssDecl.getPropertyValue("left");
    } else if (elem.currentStyle) {
        result = elem.currentStyle.left;
    } else if (elem.style) {
        result = elem.style.left;
    } else if (isNN4) {
        result = elem.left;

    return parseInt(result);
}

// Определение вертикальной координаты позиционируемого объекта
function getObjectTop(obj) {
    var elem = getRawObject(obj);
    var result = 0;
    if (document.defaultView) {
        var style = document.defaultView;
        var cssDecl = style.getComputedStyle(elem, "");
        result = cssDecl.getPropertyValue("top");
    } else if (elem.currentStyle) {
        result = elem.currentStyle.top;
    } else if (elem.style) {
        result = elem.style.top;
    } else if (isNN4) {
        result = elem.top;
    }
    return parseInt(result);
}

// Определение ширины отображенного на экране объекта
function getObjectWidth(obj) {
    var elem = getRawObject(obj);
    var result = 0;
    if (elem.offsetWidth) {
        result = elem.offsetWidth;
    } else if (elem.clip && elem.clip.width) {
        result = elem.clip.width;
    } else if (elem.style && elem.style.pixelWidth) {
        result = elem.style.pixelWidth;
    }
    return parseInt(result);
}

// Определение высоты отображенного на экране объекта
function getObjectHeight(obj) {
    var elem = getRawObject(obj);
    var result = 0;
    if (elem.offsetHeight) {
        result = elem.offsetHeight;
    } else if (elem.clip && elem.clip.height) {
        result = elem.clip.height;
    } else if (elem.style && elem.style.pixelHeight) {
        result = elem.style.pixelHeight;
    }
    return parseInt(result);
```

продолжение

Листинг 13.1 (продолжение)

```
// Возвращает ширину доступного в окне браузера пространства
function getInsideWindowWidth() {
    if (window.innerWidth) {
        return window.innerWidth;
    } else if (isIE6CSS) {
       // измеряем clientWidth элемента html
        return document.body.parentElement.clientWidth;
    } else if (document.body && document.body.clientWidth) {
        return document.body.clientWidth;
    }
    return 0;
}
// Возвращает высоту доступного в окне браузера пространства
function getInsideWindowHeight() {
    if (window.innerHeight) {
        return window.innerHeight;
    } else if (isIE6CSS) {
       // измеряем clientHeight элемента html
        return document.body.parentElement.clientHeight;
    } else if (document.body && document.body.clientHeight) {
        return document.body.clientHeight;
    }
    return 0;
}
```

Назначение этой библиотеки — предоставить единый интерфейс для того, что может вызвать неприятные затруднения при попытке приспособиться к трем отдельным моделям документа: IE 4, NN 4 и W3C DOM. Благодаря ей ваши сценарии могут вызывать простые, пригодные для всех вариантов использования функции, а о синтаксических и концептуальных различиях трех разных моделей библиотека позаботиться сама.

Показанная здесь библиотека содержит множество функций, которые могут использовать ваши сценарии. Вспомогательные функции, которые, скорее всего, будут вызываться напрямую, это:

- shiftTo(obj,x,y) — перемещает объект в заданную точку связанной системы координат;
- shiftBy(obj,deltaX,deltaY) — перемещает объект в его системе координат на заданное расстояние по осям x и y;
- setZIndex(obj,zOrder) — устанавливает порядок наложения объекта (Z index);
- show(obj) — делает объект видимым;
- hide(obj) — делает объект невидимым.

Все перечисленные функции требуют параметр, который задавал бы объект, над которым производятся действия. Поскольку библиотека не может предугадать, как ваши сценарии будут обращаться к объектам, все они могут принимать как полноценную ссылку на элемент, так и просто идентификатор (в виде строки). Если указать идентификатор, другая внутренняя функция библиотеки сформирует ссылку на элемент, нужную для модификации свойств стиля объекта. Это работает даже в специфическом случае Navigator 4, в котором вместо стиля элемента следует применить свойства объекта layer.

Чтобы просто передвинуть объект по заданным координатам в его системе координат, необходимо просто вызвать функцию shiftTo(), предав ей в качестве параметров ссылку на элемент и координаты нового местоположения:

```
shiftTo("moveableFeast" 340. 500):
```

Чтобы сдвинуть объект вдоль одной или двух осей, используется функция shiftTo(). Например, чтобы сдвинуть элемент на три пиксела влево и на пять вверх, следует использовать такой вызов:

```
shiftBy("moveableFeast", 3. -5):
```

В этой функции положительные значения соответствуют сдвигу влево и вниз, а отрицательные — вправо и вверх.

Конечно, ваши сценарии могут при необходимости вызывать любую из функций библиотеки. Например, две последние функции возвращают высоту и ширину области содержимого в окне браузера. Эти значения могут оказаться полезными при позиционировании элемента или при настройке его размеров в соответствии с размерами окна.

Также библиотека автоматически инициализирует обработчик события onload элемент body (по сути, это эквивалентно встраиванию обработчика в элемент body). Если же ваша страница сама задает обработчик в теге <body>, это назначение отменяет инициализатор, назначенный библиотекой. Таким образом, ваша собственная процедура инициализации в этом случае обязана самостоятельно вызывать функцию initDHTMLAPI(), формирующую некоторые глобальные переменные, необходимые для работы других функций библиотеки.

Как и во многих других библиотеках сценариев, вы можете убрать неиспользуемые функции и даже встроить библиотеку в код страницы. Загрузка большой библиотеки, из которой используются только несколько функций, лишь напрасно снижает пропускную способность. Нужно лишь постараться не удалять вспомогательные функции, необходимые тем функциям, которые вы вызываете. Например, если вы собираетесь применить shiftBy(), то вам необходимо оставить initDHTMLAPI(), getObject(), getRawObject(), getObjectLeft(), getObjectTop().

Если предполагаемая целевая аудитория использует только браузеры с поддержкой свойства style для всех элементов (то есть IE 4 и старше, а также Netscape 6 и старше), позиционирование объектов можно упростить, не применяя библиотеку. В качестве примера загляните в код функции shiftTo(), меняющей положение элемента, присваивая координаты (с указанием единицы измерения длины) свойствам style.left и style.top. Данная методика работает только с позиционируемыми объектами. Обычно браузеры стремятся не отображать содержимое до тех пор, пока не завершится процесс исполнения кода. Поэтому, даже если производится изменение положения по двум координатам, пользователь увидит только один прыжок из исходной точки в конечную. Для того чтобы создать анимацию, необходимо задействовать механизм setInterval(), чтобы заставить браузер перерисовывать объект (см. рецепты 13.9 и 13.10).

Смотрите также

Большая часть рецептов этой главы, а также многие другие рецепты в этой книге используют приведенную библиотеку.

13.4. Выбор между div и span

NN4 **IE4**

Задача

Необходимо выбрать оптимальный контейнер для подвижного блока.

Решение

Любой контейнер, для которого задано фиксированное или абсолютное позиционирование, становится блочным элементом. Поэтому в большинстве браузеров нет большой разницы, какой из контейнеров применять для расположения содержимого. Тем не менее использование div для позиционирования может помочь нам определить, какой из фрагментов кода относится к позиционируемому элементу.

Однако существенное отличие возникает, если для задания системы координат используется относительно позиционируемый строчный элемент. В следующем примере точка в конце абзаца сделана относительно позиционируемым элементом, относительно которого располагается другой, абсолютно позиционируемый блок:

```
<p>Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim adminim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat<span id="someSpan" style="position:relative">.
<span style="position:absolute; top:20px; left:0px; width.80px"
id="anotherSpan">
- Greek text in Latin.
</span>
</span>
</p>
```

Обсуждение

На рис. 13.2 показан внешний вид этого решения в двух окнах разного размера, чтобы показать, как абсолютно позиционируемая надпись размещается относительно точки в конце абзаца

По умолчанию любой блочный элемент HTML отображается на новой строке, следующие за ним элементы также отображаются с новой строки. Поэтому, например, стандартный элемент p всегда начинается с новой строки (а его содержимое в системах с направлением письма слева направо выровнено по левому краю). Строчные элементы (такие как span или em) не оказывают влияния на размещение содержимого. -

Если в качестве контейнера для абсолютно или фиксировано позиционированного фрагмента используется span, его содержимое полностью исключается из основного процесса отображения и помещается в отдельный блок. Но его

CSS-свойство display остается равным inline, несмотря на изрядную «блочность» подвижного элемента.

Меняя значение свойства style.display, можно полностью контролировать характеристики размещения элемента, перекрывая его поведение по умолчанию. Обычно этот подход применяется как средство вставлять и удалять элементы из страницы. С его помощью можно превратить типичный строчный элемент в блочный, присвоив свойству style.display значение block.

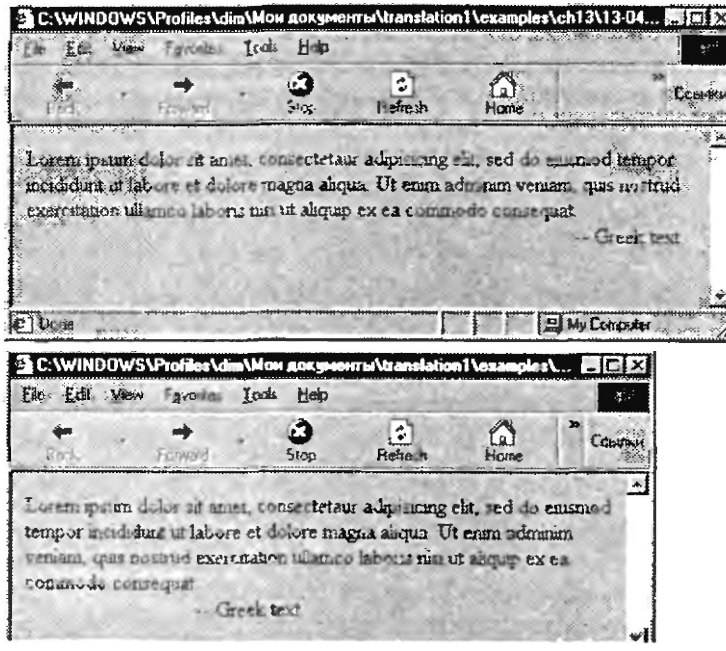


Рис. 13.2. Абсолютно позиционируемая надпись внутри относительно размещенного span

Смотрите также

В рецепте 13.2 показывается, как использовать относительно размещаемый контейнер для задания системы координат.

13.5. Управление порядком наложения (z-order)

NN4 **IE4**

Задача

Необходимо управлять тем, будет ли перекрывающийся элемент расположен на переднем или на заднем плане.

Решение

Если нужна обратная совместимость, можно применить функцию setZIndex() из DHTML-библиотеки (см. рецепт 13.3) или, если браузер поддерживает свойство style, можно напрямую менять значение свойства style.zIndex:

```
document.getElementById("mylayer").style.zIndex = "100";
```

Обсуждение

Элемент, сделанный подвижным, автоматически оказывается на слое над основным документом. Если не задать ему фоновый цвет или рисунок, по умолчанию задний фон станет прозрачным и сквозь него будет видно находящееся ниже содержимое.

Если при взаимодействии со страницей порядок наложения блоков меняться не должен, его можно просто задать изначально с помощью CSS-свойства z-index, и на этом задача будет решена. Но существуют ситуации, когда порядок наложения нужно менять. Например, на странице с несколькими перетаскиваемыми элементами (см. рецепт 13.11) нужно гарантировать, что перетаскиваемый в данный момент объект находится над всеми остальными. В противном случае при перетаскивании он может нырять под другие элементы, запутывая пользователя. Отпустив его под другими блоками, можно вообще потерять элемент.

Правила наложения довольно просты. Если два элемента имеют одинаковые значения z-index, сверху будет отображен тот из них, который объявлен в коде позже. По умолчанию значение этого свойства у всех элементов равно auto, что эквивалентно 0. Элементы с большим значением этого свойства располагаются над элементами с меньшим, благодаря чему можно управлять порядком наложения, невзирая на порядок объявления в коде, но даже отрицательное значение не может разместить объект под основной страницей.

В только что упомянутом сценарии, реализующем перетаскивание (полностью он описан в рецепте 13.11), в ответ на выбор объекта мышью значение его свойства z-index должно быть увеличено настолько, чтобы гарантировать, что все остальные объекты имеют меньшие значения. Поскольку значения, которые можно присваивать этому свойству, практически не ограничены, для элементов первого плана не так уж редко используются довольно большие числа, например 100 или 1000. После того как пользователь прекратит перетаскивание, значение z-index нужно восстановить, поэтому предварительно его нужно сохранять в переменной или в отдельном свойстве. Это необходимо, чтобы освободить на верхнем уровне место для следующего перетаскиваемого элемента.

Остерегайтесь смешения позиционируемых элементов и элементов форм. Во всех браузерах, кроме самых новых, вы можете столкнуться с проблемами, если слой должен перекрывать элемент формы. Чаще всего проблемы вызывают текстовые элементы управления: текстовые input, textarea и элементы select (у последних для отображения содержимого часто используется текстовое поле).

К сожалению, не существует способов решения этой проблемы, даже при помещении в свойство z-index самых больших значений ничего не изменить. Если переработать интерфейс и обойти проблему невозможно, единственный выход — скрывать элементы формы, когда виден подвижный слой. Самый простой спо-

соб сделать это — обернуть форму относительно позиционируемым span, а затем при необходимости менять его свойство style.visibility. При этом остальная часть страницы сдвигаться не будет.

Смотрите также

Смотрите библиотеку в рецепте 13.3, позволяющую управлять порядком наложения и решать прочие задачи DHTML.

13.6. Как расположить один элемент по центру другого

NN6 **IE5**

Задача

Разместить объект так, чтобы он был вертикально и горизонтально центрирован относительно элемента в основном документе.

Решение

Ниже приведена функция centerOnElement(), которой требуется два аргумента: идентификатор элемента в основном документе и идентификатор подвижного объекта, который следует центрировать. Функция совместима с браузерами, поддерживающими синтаксис W3C DOM обращения к элементам:

```
function centerOnElement(baseElemID, posElemID) {
    baseElem = document.getElementById(baseElemID);
    posElem = document.getElementById(posElemID);

    var offsetTrail = baseElem;
    var offsetLeft = 0;
    var offsetTop = 0;
    // учитываем режим CSS-совместимости в IE 6
    while (offsetTrail) {
        offsetLeft += offsetTrail.offsetLeft;
        offsetTop += offsetTrail.offsetTop;
        offsetTrail = offsetTrail.offsetParent;
    }
    if (navigator.userAgent.indexOf("Mac") != -1 &&
        typeof document.body.leftMargin != "undefined") {
        offsetLeft += document.body.leftMargin;
        offsetTop += document.body.topMargin;
    }

    posElem.style.left = offsetLeft + parseInt(baseElem.offsetWidth/2)
        parseInt(posElem.offsetWidth/2) + "px"
    posElem.style.top = offsetTop + parseInt(baseElem.offsetHeight/2)
        parseInt(posElem.offsetHeight/2)+ "px"
}
```

Обсуждение

Как бы ни просто звучала эта задача, при ее решении необходимо компенсировать множество несовместимостей, имеющихся в разных версиях и режимах работы Internet Explorer. Еще одним ключевым моментом, благодаря которому функция действует, является измерение размеров как статичного, так и подвижного элементов. Если эти элементы содержат «текучее» наполнение (текст, например), их размеры могут меняться в зависимости от размеров окна браузера и размера шрифтов. В этой функции перед вычислением нового положения определяются действительные размеры обоих элементов.

Вначале функция определяет абсолютные координаты объекта в основном документе. Это итеративный процесс, в котором накапливаются значения leftOffset и rightOffset основного элемента и всех родительских по смещению элементов. Во многих браузерах достаточно знать величину смещений основного элемента, но чтобы получить точные значения в IE 6 для Windows в режиме CSS-совместимости, нужно добавлять величины родительских элементов. Для решения этой задачи элементы, связаннее свойством offsetParent, обходятся по цепочке.

Далее следует подстройка под нестандартное поведение IE для Mac. В этом браузере на абсолютное положение объекта в главном документе влияют значения правого и левого полей элемента body. К сожалению, нет гарантии, что эта подстройка будет работать в будущих версиях IE для Mac. В будущем эта ошибка может быть, а может и не быть исправлена, поэтому сейчас проверка версии ничего не даст, следует быть осторожным с будущими версиями IE для Mac.

В последней части вычислений по известным абсолютным координатам основного объекта определяются координаты подвижного. Чтобы совпадали центры элементов (а не левые верхние углы), вычисляется добавка, по каждому из измерений равная разности половины размера основного элемента и половины размера подвижного. Сложив эту добавку с координатами основного объекта, получаем абсолютные координаты центрированного позиционируемого элемента.

Также следует учитывать, что пользователь может изменить размеры окна браузера, в результате чего все содержимое, в том числе и базовый элемент, будет расположено иначе. Так как никакой взаимосвязи между элементами нет (в отличие от ситуации с абсолютным и относительным позиционированием, рассмотренной в рецепте 13.2), для повторного центрирования необходимо задать обработчик события onresize для окна (обычно он встраивается в атрибут тега <body>). Если на странице есть несколько подвижных объектов, располагаемых по центру других элементов, обработчиком onresize может быть функция, производящая последовательно несколько вызовов centerOnElement() с нужными аргументами.

Учитывайте, что использованные в функции свойства смещения не входят в стандарт W3C DOM Но в DOM Level 2 нет никаких средств получения этой важной для позиционирования (а также для многих других операций) информации. Хотя эти свойства введены Microsoft, основанные на Mozilla браузеры также поддерживают их. Так, рецепт прекрасно работает в Netscape 6 и старше.

Смотрите также

Информация о режиме CSS-совместимости в IE 6, а также о том, как задать режим работы браузера, приведена в рецепте 11.13.

13.7. Как разместить элемент по центру окна или фрейма

NN4 IE4

Задача

Необходимо разместить элемент по центру окна браузера или по центру фрейма в наборе фреймов.

Решение

Оптимальную обратную совместимость предоставляет библиотека DHTMLAPI.js из рецепта 13.3. Показанная ниже функция centerOnWindow(), основанная на этой библиотеке, работает в IE 4, Navigator 4 и более новых версиях:

```
// Центруем в текущем окне или фрейме позиционируемый элемент.
// имя которого передано в качестве параметра
function centerOnWindow(elemID) {
    // 'obj' - подвижный объект
    var obj = getRawObject(elemID);
    // положение прокрутки окна
    var scrollX = 0. scrollY = 0:
    if (document.body && typeof document.body.scroll Top != "undefined") {
        scrollX += document.body.scroll Left:
        scrollY += document.body.scrollTop:
        if (document.body.parentNode &&
            typeof document.body.parentNode.scrollTop != "undefined") {
            scrollX += document.body.parentNode.scrollLeft:
            scrollY += document.body.parentNode.scroll Top;
        }
    } else if (typ of window.pageXOffset != "undefined") {
        scrollX += window.pageXOffset;
        scrollY += window.pageYOffset;
    }
    var x = Math.round((getInsideWindowWidth()/2)
        (getObjectWidth(obj)/2)) + scrollX:
    var y = Math.round((getInsideWindowHeight()/2)
        (getObjectHeight(obj)/2)) + scrollY:
    shiftTo(obj. x. y ) :
    show(obj):
}

// Возвращает высоту доступного пространства в окне браузера
function getInsideWindowHeight() {
    if (window.innerHeight) {
        return window.innerHeight:
```

```
    } else if (isIE6CSS) {
        // измеряем clientHeight элемента html
        return document.body.parentElement.clientHeight;
    } else if (document.body && document.body.clientHeight) {
        return document.body.clientHeight;
    }
    return 0;
}


// Обход ошибки перерисовки CSS-P в Netscape 4
function handleResize() {
    if (isNN4) {
        // вызов дополнительной перерисовки
        location.reload();
    } else {
        centerOnWindow("banner");
    }
}
// Удерживаем объект по центру при прокрутке
function handleScroll() {
    centerOnWindow("banner");
}

window.onresize = handleResize;
window.onscroll = handleScroll;
```

Обсуждение

Первая сложность возникает в Internet Explorer, в котором объект window не предоставляет никаких свойств, которые прямо бы содержали информацию о размере окна, внешнем или внутреннем. Таким образом, как ключ к определению размеров необходимо использовать размеры других объектов.

Эта проблема в функциях getInsideWindowHeight() и getInsideWindowWidth() решается библиотекой DHTML API. Поскольку для определения размера по обеим осям используется один алгоритм, для маленьких документов, не столь высоких и широких, как окно, могут быть получены неправильные значения. Но для всех элементов Internet Explorer поддерживаются свойства ciientHeight и clientWidth, которые в случае body соответствуют размерам тела документа, в данный момент видимого в окне. В IE 5 и старше (в том числе в IE 6 в режиме обратной совместимости) размеры элемента body всегда равны размерам внутренней части окна браузера, даже если документ по размерам меньше. Тем не менее в режиме CSS-совместимости IE 6 размеры окна хранятся в свойствах объекта html (родительского по отношению к body). Поэтому в браузерах, не поддерживающих имеющееся в Netscape свойство window.innerHeight, две сходные API-функции определяют размеры окна по наиболее ясному источнику. Например, так в рецепте 13.3 определяется высота окна:

```
// Возвращает высоту доступного пространства в окне браузера
function getInsideWindowHeight() {
    if (window.innerHeight) {
        return window.innerHeight;
```

```
    } else if (isIE6CSS) {
        // Измеряем значение clientHeight элемента html
        return document.body.parentElement.clientHeight;
    } else if (document.body && document.body.clientHeight) {
        return document.body.clientHeight;
    }
    return 0;
}
```

Другой фактор, который необходимо учитывать, — это прокрутка документа по вертикальной и горизонтальной осям. Например, если пользователь прокрутит окно вниз, точка с координатами 0,0 передвинется вверх, а следовательно, вертикальная координата центрируемого элемента должна быть увеличена на соответствующее количество пикселов.

W3C DOM Level 2 не предоставляет свойств для доступа к такой информации, как прокрутка страницы, а также ко многим другим свойствам окон. В браузерах четвертой версии Microsoft и Netscape реализовали собственные способы доступа к положению прокрутки окна. Хотя в Netscape 7 из соображений удобства реализована поддержка введенных Microsoft свойств (scrollLeft и scrollTop), надежнее использовать ветвление, чтобы определять прокрутку наиболее совместимым способом. Большая часть строк в коде функции centerOnWinow() из этого примера ответственна за считывание этих значений, чтобы затем применить их к координатам, передаваемых в DHTML API-функцию shiftTo().

Элемент, размещенный однажды по центру, не останется там, если пользователь прокрутит страницу или изменит размеры окна. Чтобы компенсировать эти вполне обычные действия, необходимо задать обработчики событий, реагирующие как на изменение размера, так и на прокрутку. Возможность обработать изменение размера окна поддерживается всеми браузерами, способными использовать библиотеку DHTML API; событие, соответствующее прокрутке, имеется только в IE 4 и Netscape 7, а также в более новых браузерах. Чтобы обработчики этих событий работали наиболее надежно, следует назначить их через свойства объекта window. В обработчик, назначенный таким образом, нельзя передавать аргументы (например, идентификаторы элементов, которые нужно удерживать по центру). Поэтому функция-обработчик должна вызывать centerOnWindow(), вручную передавая ей необходимые параметры, как в показанном ниже примере. В этом примере также применяется глобальная переменная, isNN4, значение которой устанавливается DHTML API:

```
// Обход ошибки перерисовки CSS-P в Netscape 4
function handleResize() {
    if (isNN4) {
       // вызов дополнительной перерисовки
        location.reload();
    } else {
        centerOnWindow("banner");
    }
}
// Удерживаем объект по центру при прокрутке
function handleScroll() {
    centerOnWindow("banner");
}
```

```
window onresize - nandleResize:
window onscroll - handleScroll;
```

Лучший способ зафиксировать элемент по центру окна — использование фиксированного позиционирования. При этом все равно необходимо устанавливать начальное положение объекта после загрузки страницы, так как каждое из окон браузера может иметь свой размер. К сожалению, IE вплоть до версии 6 не поддерживает этот тип CSS-позиционирования.

Смотрите также

Об управлении режимом CSS совместимости в IE 6 рассказывается в рецепте 11.13.

13.8. Определение положения обычного элемента

NN6 IE5

Задача

Выяснить координаты (в пикселах) обычного, репозиционируемого элемента на странице.

Решение

Приведенная ниже функция getElementPosition() возвращает объект, свойства left и top которого равны соответствующим абсолютным координатам объекта, а идентификатор передан в нее:

```
function getElementPosition(elemID) {
    var offsetTrail = document.getElementById(elemID);
    var offsetLeft = 0;
    var offsetTop = 0;
    while (offsetTrail) {
        offsetLeft += offsetTrail.offsetLeft;
        offsetTop += offsetTrail.offsetTop;
        offsetTrail - offsetTrail.offsetParent;
    }
    if(navigator.userAgent.indexOf("Mac") != -1 &&
        typeof document.body.leftMargin !- "undefined") {
        offsetLeft += document.body.leftMargin;
        offsetTop +- document.body.topMargin;
    }
    return {left:offsetLeft, top.offsetTop};
}
```

Эта функция совместима с браузерами, поддерживающими синтаксис обращения к элементам W3C DOM.

Обсуждение

Обычно координаты элемента страницы определятся для того, чтобы впоследствии применить их для размещения какого-либо подвижного объекта. Поскольку положение элементы страницы значительно меняется при изменении размеров окна браузера или шрифтов, координаты необходимо вычислять заново каждый раз после того, как страница загружена, изменен размер окна или его содержимое изменилось в ответ на динамическую вставку или удаление фрагментов.

Хотя некоторые браузеры сообщают искомые значения просто через свойства объекта offsetLeft и offsetTop, в других требуется добавлять смещения, которые задаются родительскими элементами, задающими систему координат (определяемыми по ссылке в свойстве offsetParent). Следовательно, функция должна содержать цикл, перебирающий для объекта, идентификатор которого передан в функцию, всю цепочку родителей и суммирующий дополнительные смещения, если таковые существуют.

Для абсолютно позиционируемых элементов в стиле CSS эта функция не нужна, потому что координаты можно определить по значению свойств style. left и style.right (или через свойство, содержащее действующее значение стиля, как продемонстрировано в рецепте 11.12).

Смотрите также

Смотрите рецепт 11.12, в котором показано, как можно прочитать начальные значения свойств стилей, заданных с помощью тегов <style> или <link>.

13.9. Прямолинейная анимация

NN6 IE5

Задача

Анимировать элемент так, чтобы он переместился из одной точки страницы в другую по прямому пути.

Решение

Чтобы анимировать движение объекта по прямолинейному пути можно подключить к странице библиотеку animeLine.js и вызвать из нее функцию initSLAnime(), передав ей в следующей последовательности как минимум пять первых параметров:

- идентификатор подвижного элемента (в виде строки);
- координата x исходной позиции;
- координата y исходной позиции;
- координата x конечной позиции;
- координата y конечной позиции.

Например:

```
initSLAnime("floater", 100, 100, 400, 360);
```

Обсуждение

Хотя для того, чтобы передвинуть элемент вдоль заданной прямой линии, можно обойтись гораздо меньшим количеством кода, чем показано в этом рецепте, показанная в листинге 13.2 библиотека **animeLine.js** дает более общее решение. Для функции из этой библиотеки достаточно указать идентификатор элемента, его начальное и конечное положение, а также относительную скорость перемещения. Все вычисления, необходимые для решения этой задачи, производятся внутри библиотеки и не зависят от длины или длительности перемещения.

Листинг 13.2. Библиотека animeLine.js для прямолинейной анимации

```
// объект, хранящий многочисленные параметры, относящиеся к движению
var anime;

// инициализация объекта anime по умолчанию
function initAnime() {
    anime = {elemID:"",
             xCurr:0,
             yCurr:0,
             xTarg:0,
             yTarg:0,
             xStep:0,
             yStep:0,
             xDelta:0,
             yDelta:0,
             xTravel:0,
             yTravel:0,
             vel:1,
             pathLen:1,
             interval:null
            }

// заполнение анимационного объекта необходимыми заданными и вычисленными
// значениями
function initSLAnime(elemID, startX, startY, endX, endY, speed) {
    initAnimeO;
    anime.elemID = elemID;
    anime.xCurr = startX;
    anime.yCurr = startY;
    anime.xTarg = endX;
    anime.yTarg = endY;
    anime.xDelta = Math.abs(endX - startX);
    anime.yDelta = Math.abs(endY - startY);
    anime.vel = (speed) ? speed : 1;
    // установка начального положения элемента
    document.getElementById(elemID).style.left = startX + "px";
    document.getElementById(elemID).style.top = startY + "px";
    // длина линии между началом и концом
    anime.pathLen = Math.sqrt((Math.pow((startX - endX), 2)) +
    (Math.pow((startY - endY), 2)));
    // длина отдельных шагов вдоль каждой из осей
    anime.xStep = parseInt(((anime.xTarg - anime.xCurr) / anime.pathLen) *
```

```
        anime.vel):
    anime.yStep - parseInt(((anime.yTarg - anime.yCurr) / amme pathLen) *
        anime.vel):
    // запускаем последовательный вызов функции анимации
    anime.interval - setInterval("doSLAnimation()"  10):
}

// вычисление следующего шага и установка значений свойств стиля
function doSLAnimation() {
    if ((anime.xTravel + anime.xStep) <= anime.xDelta &&
        (anime.yTravel + anime.yStep) <= anime.yDelta) {
        var x - anime.xCurr + anime.xStep;
        var y = anime.yCurr + anime.yStep:
        document.getElementById(anime.elemID).style.left - x + "px",
        document.getElementById(anime.elemID).style.top = y + "px":
        anime.xTravel += Math.abs(anime.xStep);
        anime.yTravel +- Math.abs(anime.yStep);
        anime.xCurr = x:
        anime.yCurr = y:
    } else {
        document.getElementById(anime.elemID).style.left - anime.xTarg +
            "px":
        document.getElementById(anime.elemID).style.top = anime.yTarg +
            •px":
        clearInterval(amme.interval):
    }
}
```

В начале библиотеки задается пустой объект anime, который затем будет хранить информацию для каждой вызванной вами анимации. При каждом вызове initSLAnime() свойства этого объекта заполняются нулевыми начальными значениями, а затем в них помещаются переданные в виде параметров и вычисляемые из них величины. Последним шагом инициализации является запуск процесса прямолинейной анимации с помощью функции **setInterval(),** которая многократно вызывает подпрограмму анимации. Идентификатор запущенного таймера также сохраняется в одном из свойств объекта anime, вместо того чтобы помещать его в глобальную переменную, замусоривающую документ.

При каждом вызове функция doSLAnimationO продвигает объект к конечной точке до тех пор, пока он ее не достигнет. Когда объект достигнет последней точки, он позиционируется строго по ее координатам, а таймер отключается, чтобы остановить последующие вызовы.

Все координаты здесь измеряются в абсолютной системе координат, как те координаты, в соответствии с которыми располагается объект в начале и конце пути. Если вам нужно привязать начальную и/или конечную точку пути к содержимому страницы, можно применить для определения абсолютных координат объектов на странице рецепт 13.8, а затем передать эти значения в initSLAnime(). Прямолинейная траектория может располагаться под любым углом (в том числе строго вертикально или строго горизонтально) и вести в любом направлении. Следует осторожно использовать координаты за пределом окна браузера, так как исчезнувший из поля зрения объект легко потерять.

Если необходимо перемещать объект вдоль более сложной траектории, можно последовательно связать несколько вызовов initSLAnime(), но в этом случае перед следующим вызовом нужно обязательно дать завершиться предыдущему. Самый простой способ — добавить в объект anime новое свойство, скажем, next, и инициализировать его нулем. Так как из другого места сценария может неоднократно вызываться функция initAnime(), значение нового свойства нужно сохранять между вызовами. Это можно реализовать, добавив в код инициализации такую строку:

```
next: (anime.next) ? anime.next : 0.
```

Далее, следует создать управляющую функцию, использующую массив координат вершин точек, составляющих сложную траекторию. Например, такая функция использует три прямых отрезка, составляющих треугольник:

```
function animatePolygon(elemID) {
    // подготовка объекта anime для следующего этапа
    initAnime();
    // формирование массива координат
    var coords - new Array()
    coords[0] - [200. 200. 100. 400]:
    coords[1] - [100. 400. 300. 400]:
    coords[2] = [300. 400. 200. 200]:
    // основываясь на значении anime.next. передаем следующую группу
    // координат.
    if (anime.next < coords.length) {
        initSLAnime(elemID. coords[anime.next][0]. coords[anime.next][1].
        coords[amme.next][2]. coords[anime.next][3]. 10).
       // переход к следующему отрезку
        anime.next++·
    } else {
       // после завершения сбрасываем счетчик 'next'
        anime.next - 0:
    }
}
```

Еще одно изменение, которое нужно сделать, — передвинуть вызов initAnimeO в новую функцию animatePolygon(). Наконец, в doSLAnimetion() для перехода к новому этапу после того, как завершен предыдущий, следует опять вызывать функцию animatePolygon(), передавая ей идентификатор подвижного элемента:

```
animatePolygon(anime.elemID):
```

Вы можете использовать для анимации столько точек, сколько вам нужно, при этом возвращаться в исходное положение не обязательно.

Необязательный шестой параметр функции initSLAnime() характеризует скорость перемещения, которая определяется величиной скачка между отдельными шагами анимации. По умолчанию он равен 1, что довольно медленно. Пользователи могут захотеть увидеть более живое перемещение, для которого подойдет значение 10. Но никто не мешает вам экспериментировать с другими значения-

ми в поисках оптимальной скорости. Также можно экспериментировать со вторым аргументом функции setInterval() (например, передавая большие значения, чем приведенные в решении), чтобы перемещаться еще медленнее.

Следует учитывать, что видимая скорость перемещения не будет одинаковой в разных браузерах. Этот тип анимации отличается от анимации в специализированных средах, таких как Flash. Это является той ценой, которую приходится платить за то, что в движении могут участвовать стандартные элементы HTML, способные перемещаться по всей странице и не ограниченные прямоугольником, как Flash.

Смотрите также

Как определить координаты элемента страницы, рассказывается в рецепте 13.8. Создание объектов продемонстрировано в рецепте 3.7. Анимация по круговой траектории продемонстрирована в рецепте 13.10.

13.10. Анимация по кругу

NN6 IE5

Задача

Заставить объект двигаться по кругу.

Решение

Чтобы добиться нужного эффекта, присоедините к странице библиотеку animeCirc.js (листинг 13.3 в обсуждении) и вызовите функцию initCircAnime(), передав ей параметры в следующем порядке:

1. Идентификатор подвижного объекта (строка).
2. Координата x начальной и конечной точки траектории.
3. Координата y начальной и конечной точки траектории.
4. Четное целое число, задающее число точек траектории.
5. Целое число, задающее относительный радиус окружности.

Типичный набор параметров может выглядеть так:

```
initCircAnuime("rounder", 200, 200, 36, 10);
```

Обсуждение

Процесс круговой анимации во многом похож на прямолинейную из рецепта 13.9, но при описании точек траектории используется тригонометрия. В листинге 13.3 показана библиотека animeCircjs, код которой решает эту задачу.

Листинг 13.3. Библиотека animeCirc.js для перемещения объекта по кругу

```
// объект, содержащий многочисленные характеристики анимации
var anime - new Object O:

// инициализация по умолчанию объекта anime
function initAnime() {
    anime - {elemID:"".
             xStart:0,
             yStart:0,
             xCurr:0.
             yCurr:0,
             next:1,
             pts:1,
             radius:1,
             interval :null
            }:
}

// заполняем объект anime различными переданными и вычисленными значениями
function initCircAnime(elemID, startX, startY. pts. radius) {
    initAnimeO:
    anime.elemID - elemID;
    anime.xCurr » anime.xStart - startX:
    anime.yCurr - anime.yStart - startY:
    anime.pts - pts:
    anime.radius - radius:
    // задание начального положения элемента
    document.getElementById(elemID).style.left - startX + "px".
    document.getElementById(elemID).style.top - startY + "px":
    // запуск многократного вызова анимации
    anime.interval - setInterval("doCircAnimation()", 10):
}

function doCircAnimation() {
    if (anime.next < anime.pts) {
        var x - anime.xCurr +
           Math.round(Math.cos(anime.next * (Math.PI/(anime.pts/2))) *
              anime.radius):
        var y - anime.yCurr +
           Math.round(Math.sin(anime.next * (Math.PI/(anime.pts/2))) *
              anime.radius):
        document.getElementById(anime.elemID). style.left - x + "px":
        document.getElementById(anime.elemID)  style.top = y + "px":
        anime.xCurr = x:
        anime.yCurr - y;
        anime.next++;
    } else {
        document.getElementById(anime.elemID).style.left - anime.xStart +
           "px";
        document.getElementById(anime.elemID).style.top - anime.yStart +
           "px":
        clearInterval(anime.interval):
    }
```

Библиотека начинается с определения абстрактного объекта, инициализирующегося каждый раз, когда начинается новая анимация. Сценарии должны вызы-

вать функцию initCircAnime(), присваивающую значения свойствам этого объекта. В конце библиотеки расположена функция, вызываемая при каждом шаге анимации в ответ на setInterval().

Плавность перемещения определяется числом точек в траектории, в которых будет обновляться изображение на экране. Это число становится верхней границей для значения anime.next в функции doCircAnimation(). Чтобы завершить полный круг, число, на которое делится PI в следующих двух строках, должно быть равно половине этого максимального значения. Радиус окружности задается множителем в конце двух этих строк. Это значение (хранящееся в anime.radius) представляет собой не пиксельную меру, а скорее множитель, определяющий радиус. Чем больше это число, тем больше радиус.

Если помещать в anime.pts большие значения (например, 72), движение становится плавнее, так как дуги между точками уменьшаются. Это также приводит к замедлению движения, поскольку интервал между шагами анимации (10) практически минимален. С другой стороны, хотя траектория со слишком маленьким числом точек и быстрее, пользователям она может показаться слишком отрывистой.

Смотрите также

Прямолинейная анимация описывается в рецепте 13.9. О создании объектов рассказывается в рецепте 3.7.

13.11. Создание перетаскиваемых элементов

NN4 IE4

Задача

Необходимо сделать так, чтобы пользователь мог перетащить элемент с одного места страницы на другое.

Решение

Чтобы дать пользователю возможность перетаскивать изображения, используйте две библиотеки: dragImg.js с листинга 13.4 в обсуждении и DHTML API из рецепта 13.3. Библиотека draglmg.js создана в расчете на то, что перетаскиваться будет изображение, помещенное внутрь элемента div, атрибут class которого будет иметь значение draggable. Кроме того, идентификатор div должен состоять из идентификатора вложенного в него изображения с приписанными в конце буквами Wrap. Вот пример кода, задающего два перетаскиваемых изображения:

```
<div id="imgAWrap" class="draggable"><img id="imgA" name="imgA"
src="ora1.gif" width="230" height="102" border="0"
alt="Первый перетаскиваемый объект"></div>
<div id="imgBWrap" class="draggable"><img id="imgB" name="imgB"
src="ora2.jpg" width="281" height="100" border="0"
alt="Второй перетаскиваемый объект"></div>
```

В действительности эта библиотека двигает элементы **div**, но события мыши, запускающие процесс перетаскивания и контролирующие его, должны передаваться во вложенные **img**.

С помощью таблиц стилей необходимо определить эти div как позиционируемые объекты. В качестве примера далее приведено определении стилей для двух показанных выше элементов:

```
<style type="text/css">
  #imgAWrap {position:absolute; left:35%; top:20%; width:230px;
             height:102px; border:solid black 1px; z-index:0}
  #imgBWrap {position:absolute; left:40%; top:25%; width:281px;
             height:100px; border:solid black 1px; z-index:0}
</style>
```

Прочая инициализация и назначение обработчиков событий происходят в двух функциях, вызываемых событием onload тела документа, одна из них — для библиотеки DHTML API, а другая — для **dragImg.js**:

```
<body onload="initDHTMLAPI(); initDrag()">
```

Обсуждение

В листинге 13.4 приведен код библиотеки draglmg.js. Ее можно подключить к странице после библиотеки DHTML API с помощью таких строк:

```
<script language="JavaScript" type="text/javascript"
   src="../js/DHTMLAPI.js"></script>
<script language="JavaScript" type="text/javascript"
   src="dragImg.js"></script>
```

Библиотека поддерживает любое количество перетаскиваемых изображений и не конфликтует с неподвижными.

Листинг 13.4. Библиотека draglmg.js

```
// Глобальная переменная со ссылкой на выбранный элемент
var selectedObj;

// Глобальная переменная, сохраняющая относительные координаты щелчка мыши
var offsetX, offsetY;

// Задать глобальные ссылки на выделенный и перетаскиваемый элемент
function setSelectedElem(evt) {
    var target = (evt.target) ? evt.target : evt.srcElement;
    var divID = (target.name && target.src) ? target.name + "Wrap" : "";
    if (divID) {
        if (document.layers) {
            selectedObj = document.layers[divID];
        } else if (document.all) {
            selectedObj = document.all(divID);
        } else if (document.getElementById) {
            selectedObj = document.getElementById(divID);
        }
        setZIndex(selectedObj, 100);
        return;
    }
```

```
    selectedObj = null;
    return;
}

// "Включаем" выделенный элемент
function engage(evt) {
    evt - (evt) ? evt : event:
    setSelectedElem(evt);
    if (selectedObj) {
        if (document.body && document.body.setCapture) {
          // включаем перехват событий в IE/Win
            document.body.setCapture();
        }
        if (evt.pageX) {
            offsetX = evt.pageX - ((selectedObj.offsetLeft) ?
                      selectedObj.offsetLeft : selectedObj.left);
            offsetY - evt.pageY - ((selectedObj.offsetTop) ?
                      selectedObj.offsetTop : selectedObj.top);
        } else if (typeof evt.offsetX != "undefined") {
            offsetX = evt.offsetx - ((evt.offsetx < -2) ?
                      0 : document.body.scroll Left);
            offsetX -- (document.body.parentElement &&
                      document.body.parentElement.scrollLeft) ?
                      document.body.parentElement.scrollLeft • 0
            offsetY = evt.offsetY - ((evt.offsetY< -2) ?•
                      0 : document body.scrollTop);
            offsetY -- (document.body.parentElement &&
                      document.body.parentElement.scrollTop) ?
                      document.body.parentElement.scrollTop • 0
        } else if (typeof evt.clientX != "undefined") {
            offsetX - evt.clientX - ((selectedObj.offsetLeft) ?
                      selectedObj.offsetLeft . 0 ):
            offsetY = evt.clientY - ((selectedObj.offsetTop) ?
                      selectedObj.offsetTop : 0 ):
        }
        return false:
    }
}


// Перетаскивание элемента
function dragIt(evt) {
    evt = (evt) ? evt : event:
    if (selectedObj) {
        if (evt.pageX) {
            shiftTo(selectedObj. (evt.pageX - offsetX). (evt.pageY -
              offsetY)):
        } else if (evt.clientX j| evt.clientY) {
            shiftTo(selectedObj, (evt.clientX - offsetX). (evt.clientY -
              offsetY)):
        }
        evt.cancelBubble = true:
        return false:
    }
}
```

продолжение ⇙

Листинг 13.4 (продолжение)

```
// "Выключаем" выбранный элемент
function release(evt) {
    if (selectedObj) {
        setZIndex(selectedObj, 0):
        if (document.body && document.body.releaseCapture) {
           // прекращение перехвата событий в IE/Win
            document.body.releaseCapture():
        }
        selectedObj - null;
    }
}

// Назначение обработчиков событий в Navigator и в IE
function initDrag0 {
    if (document.layers) {
       // Включаем перехват этих событий в модели NN4
        document.captureEvents(Event.MOUSEDDWN | Event.MOUSEMOVE |
           Event.MOUSEUP):
        return:
    } else if (document.body & document.body.addEventListener) {
       // Включаем перехват этих событий в модели W3C DOM
       document.addEventListener("mousedown", engage, true);
       document.addEventListener("mousemove", dragIt, true):
       document.addEventListener("mouseup", release, true);
       return;
    }
    document.onmousedown - engage;
    document.onmousemove - draglt:
    document.onmouseup » release:
    return:
}
```

В начале библиотеки объявляется несколько глобальных переменных, необходимых для передачи информации между началом процесса перетаскивания и самим перетаскиванием. Одна переменная, selectedObj, содержит ссылку на перетаскиваемый объект. Значения пары переменных offsetX и offsetY устанавливаются функцией engage() и постоянно используются при позиционировании в процессе перетаскивания.

Функция setSelectedElem(), вызываемая из engage(), проверяет информацию, передаваемую с каждым событием mousedown на странице, и определяет, относится ли событие к элементу img (по наличию в них свойства src). Если это так, ссылка на элемент помещается в глобальную переменную selectedObject. Если вести разработку в строгом соответствии с W3C DOM, лучше определять элемент по имени класса, доступном через свойство className (например, так: target.className == "draggable").

Функция engage() является первым этапом обработки событий mousedown. После вызова setSelectedElement() в IE для Windows включается механизм перехвата событий (чтобы увеличить производительность процесса обработки событий). Затем вычисляется и сохраняется смещение между координатами события и левым верхним углом объекта. Это позволяет затем удерживать положение элемента относительно указателя мыши. Условные операторы здесь не-

обходимы для того, чтобы приспособиться к широкому диапазону браузеров, включая IE 4 для Macintosh (который ведет себя иначе, чем IE 4 для Windows), Navigator 4 и IE 6 в режиме CSS-совместимости (относящиеся к прокрутке свойства document.body.parentElement).

Функция, которая действительно перемещает объект (в ответ на событие mousemove), — это dragIt(). Она вычисляет новое положение элемента, используя сохраненные значения смещения элемента относительно курсора. Затем с помощью DHTML DOM функция shiftTo() устанавливает новое текущее положение элемента. Когда пользователь отпускает кнопку мыши, функция release() восстанавливает исходный порядок наложения элементов, а глобальная переменная selectedObj обнуляется.

Функция инициализации initDrag() в зависимости от поддерживаемой браузером модели событий устанавливает ряд обработчиков событий. Поскольку эта библиотека поддерживает совместимость в том числе и с моделью событий Netscape 4, одна из ветвей кода включает в этом браузере режим перехвата событий. В браузерах, поддерживающих модель событий W3C DOM (например, в семействе Mozilla), обработчики событий назначаются с помощью метода addEventListener(), которому в третьем параметре передается значение true, чтобы включить перехват событий. Во всех остальных браузерах, включая IE, имеются три обработчика событий на уровне объекта document.

То, что обработчики событий назначаются на уровне документа, а не для самих элементов div, может показаться странным. Причина в том, что пользователь может перетаскивать объект быстрее, чем браузер будет обновлять его положение. Когда указатель мыши покидает прямоугольную область элемента, событие mousemove уже не передается объектам div или img. То же касается и события mouseup, если оно произойдет вне элемента. Так может возникнуть ситуация, когда элемент считает, что он выделен, но не может реагировать на перемещение мыши до тех пор, пока указатель не вернется на него. Размещая обработчики на уровне документа, можно гарантировать, что события будут обработаны, даже если произошли не над элементом. В большинстве браузеров библиотека использует соответствующий вариант перехвата событий. В остальных, например в IE для Macintosh, используется механизм распространения событий, когда сначала они появляются на уровне подвижного объекта, а затем передаются на уровень документа для обработки.

Если необходимо ограничить область, в которой разрешено перетаскивать элемент, можно задать прямоугольную зону и удерживать объект в ней. Для этого сначала следует определить глобальный объект, содержащий координаты области:

```
var zone - {left:20. top:20. right:400. bottom:400}:
```

Затем нужно модифицировать функцию dragIt() из рецепта 13.4, чтобы она не позволяла переместить элемент за пределы зоны:

```
function dragIt(evt) {
    evt = (evt) ? evt : event:
    var x. y. width. height:
    if (selectedObj) {
        if (evt.pageX) {
            x = evt.pageX - offsetX:
```

```
            y = evt.pageY - offsetY;
        } else if (evt.clientX || evt.clientY) {
            x = evt.clientX - offsetX;
            y = evt.clientY - offsetY;
        }
        width = getObjectWidth(selectedObj);
        height = getObjectHeight(selectedObj);
        x = (x < zone.left) ? zone.left :
            ((x + width > zone.right) ? zone.right - width  x);
        y = (y < zone.top) ? zone.top :
            ((y + height > zone.bottom) ? zone.bottom  height  y);
        shiftTo(selectedObj, x, y);
        evt.cancelBubble = true;
        return false;
    }
}
```

Эта модификация использует функции DHTML API, позволяющие легко определить высоту и ширину подвижного объекта. Перед перемещением новые координаты проверяются на попадание в область. Если координата не попадает, ее значение устанавливается равным ближайшему подходящему числу. Это позволяет перетаскивать элемент вдоль одной из осей, если он достиг другой.

Смотрите также

Используемая библиотека DHTML API описана в рецепте 13.3. Замечания о режиме CSS-совместимости IE 6 смотрите в рецепте 11.13.

3.12. Прокрутка содержимого div

NN7 IE5

Задача

Необходимо дать пользователям возможность прокручивать вверх и вниз содержимое отдельной области на странице, не прибегая к системным полосам прокрутки.

Решение

В этом решении как сам контейнер с прокручиваемым содержимым, так и кнопки, управляющие прокруткой, являются элементами HTML. HTML-код для этого примера вы можете увидеть в листинге 13.5, приведенном в обсуждении. Базовый сценарий для управления прокручиваемыми областями на странице расположен в библиотеке scrollButtons.js, код которой приведен в листинге 13.6.

Чтобы применить этот рецепт, к странице нужно подключить и инициализировать две библиотеки: **DHTMLAPI.js** из рецепта 13.3 и scrollButtons.js. Инициализация должна происходить в обработчике события onload тела страницы:

```
<body onload="initDHTMLAPI(); initScrollers()">
```

Функция initScrollers() не обязательно должна быть частью библиотеки, так как она должна содержать подробности о каждой области прокрутки на странице. Например, показанная ниже функция создает объект JavaScript, управляющий одной прокручиваемой областью:

```
function initScrollers() {
    scrollBars[0] - new scrollBar("outerWrapper0". "innerWrapper0".
       "lineup0". "linedown0"):
    scrollBars[0].initScroll():
}
```

Обсуждение

Основная часть HTML-кода для этого рецепта приведена в листинге 13.5. Прокручиваемая область сформирована из нескольких вложенных элементов div. Содержимое расположено в паре вложенных контейнеров. Из них только внешний определяет прямоугольную область видимости, а само содержимое помещено во внутренний, innerWrapper0. Числа на конце идентификаторов в этом примере помогают проиллюстрировать возможность использовать несколько прокручиваемых областей на одной странице и избежать конфликта имен, назначая компонентам уникальные идентификаторами.

Листинг 13.5. HTML-код прокручиваемой области и управляющего элемента

```
<div id="pseudoWindow0" style="position:absolute; top:200px; left:45%">
<div id="outerWrapper0" style="position:absolute; top:0px; left:0px;
    neight:150px; width:100px; overflow:hidden; border-top:4px solid #666666;
    border-left 4px solid #666666· border-right:4px solid #cccccc;
    border-bottom:4px solid #cccccc; background-color:#ffffff">
<div id="innerWrapper0" style="position:absolute  top:0px; left:0px; padding 5px.
    font:10px Arial, Helvetica. sans-serif">
<p style="margin-top:0em"> Lorem ipsum dolor sit amet. consectetaur
...</p>


</div>
</div>
<img id="lineup0" class="lineup" src="scrollUp.gif" height="16" width="16"
    alt="Scroll Up" style="position:absolute; top:10px; left:112px" />
<img id="linedown0" class="linedown" src="scrollDn.gif" height="16"
   width="16" alt="Scroll Down"
    style="position:absolute; top:128px; left:112px" />
</div>
```

Кнопки управления прокруткой (в данном случае — по вертикали) представляют собой простые изображения. Изображения стрелок абсолютно позиционированы в системе координат внешнего div (pseudoWindow0). Поэтому, если для каких-либо других целей внешний контейнер нужно будет передвинуть, кнопки сохранят свое относительное положение относительно всего набора компонентов. Рисунок 13.3 иллюстрирует внешний вид прокручиваемого псевдоокна, код которого приведен в листинге 13.5.

Задача библиотеки сценариев scrollButtons.js (листинг 13.6) заключается в том, чтобы перемещать внутренний контейнер вверх и вниз. Его содержимое будет отсекаться внешним контейнером, у которого свойству overflow присвоено значение hidden.

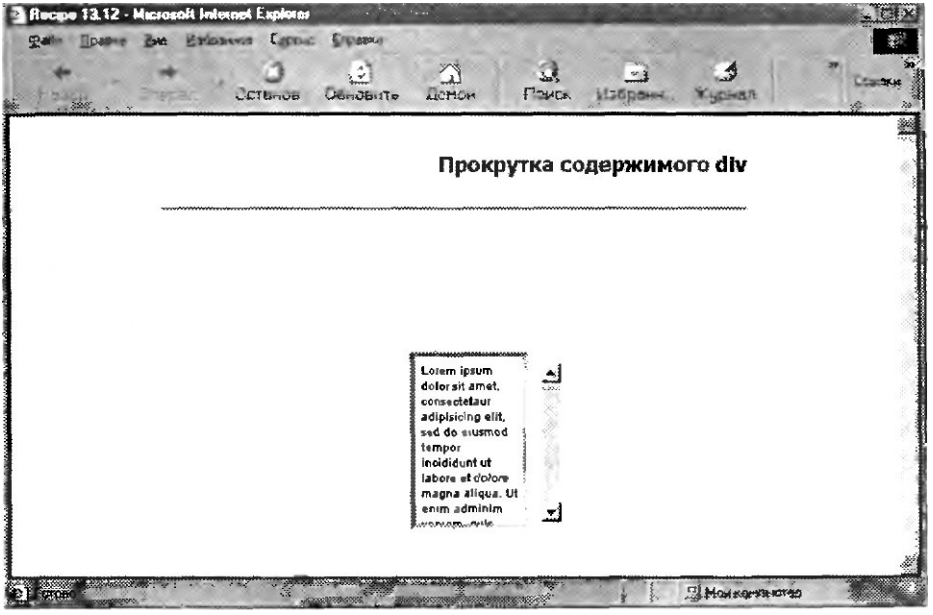


Рис. 13.3. Прокручиваемый контейнер

Листинг 13.6. Библиотека scrollButtons.js

```
// Глобальные переменные
var scrollEngaged = false;
var scrollInterval;
var scrollBars = new Array();

// Считывание действующих настроек стиля
function getElementStyle(elemID, IEStyleAttr, CSSStyleAttr) {
    var elem = document.getElementById(elemID);
    if (elem.currentStyle) {
        return elem.currentStyle[IEStyleAttr];
    } else if (window.getComputedStyle) {
        var compStyle = window.getComputedStyle(elem, "");
        return compStyle.getPropertyValue(CSSStyleAttr);
    }
    return "";
}

// Конструктор абстрактного объекта
function scrollBar(ownerID, ownerContentID, upID, dnID) {
    this.ownerID = ownerID;
    this.ownerContentID = ownerContentID;
    this.index = scrollBars.length;
```

```
    this.upButton - document.getElementById(upID);
    this.dnButton - document.getElementById(dnID);
    this.upButton.index = this.index.
    this.dnButton.index - this.index;

    this.ownerHeight - parseInt(getElementStyle(this.ownerID, "height"
       "height"));

    this.contentElem = document.getElementById(ownerContentID);
    this.contentFontSize - parseInt(getElementStyle(this.ownerContentID.
        "fontSize", "font-size"));
    this.contentScroll Height - (this.contentElem.scrollHeight) ?
        this.contentElem.scrollHeight : this.contentElem.offsetHeight;
    this.initScroll = initScroll;
}

// Назначение обработчиков событий кнопкам прокрутки
function initScroll() {
    this.upButton.onmousedown - handleScrollClick;
    this.upButton.onmouseup = handleScrollStop;
    this.upButton.oncontextmenu - blockEvent;

    this.dnButton.onmousedown = handleScrollClick;
    this.dnButton.onmouseup - handleScrollStop;
    this.dnButton.oncontextmenu - blockEvent;

    var isIEMac = (navigator.appName.indexOf("Explorer") !- -1 &&
       navigator.userAgent.indexOf("Mac") != -1);
    if (!isIEMac) {
        document.getElementById("innerWrapper0").style.overflow - "hidden";
    }


/*******************************
     Обработчики событий
*******************************/
// Выключение прокрутки
function handleScrollStop() {
    scroll Engaged = false;
}

// Блокирование контекстного меню для Mac (удержание кнопки мыши)
function blockEvent(evt) {
    evt - (evt) ? evt : event;
    evt.cancelBubble = true;
    return false;
}

// Инициация прокрутки содержимого по щелчку на кнопке
function handleScrollClick(evt) {
    var fontSize;
    evt - (evt) ? evt . event;
    var target - (evt.target) ? evt.target : evt.srcElement;
    var index = target.index;
    fontSize = scrollBars[index].contentFontSize;
```

продолжение ↵

Листинг 13.6 (продолжение)

```
    fontSize - (target.className — "lineup") ? fontSize : -fontSize:
    scroll Engaged - true
    scroll By(index. parseInt(fontSize));
    scrollInterval = setInterval("scrollBy(" + index + ". " +
        parseInt(fontSize) + ")". 100):
    evt.cancel Bubble = true:
    return false:
}

// Выполнение однократной или многократной (с помощью setInterval())
// прокрутки
function scrollBy(index. px) {
    var scroller - scrollBars[index];
    var elem - document.getElementById(scroller.ownerContentID);
    var top - parseInt(elem.style.top);
    var scrollHeight - parseInt(scroller.contentScrollHeight);
    var height - scroller.ownerHeight;
    if(scrollEngaged && top + px >- -scrollHeight + height && top + px<=0) {
        shiftBy(elem. 0. px):
    } else {
        clearInterval(scrollInterval):
    }
}
```

Формируя абстрактные объекты для хранения информации о паре кнопок прокрутки и области содержимого, библиотека реализует объектно-ориентированный подход. Это упрощает прокручивание нескольких областей одновременно. В начале библиотеки определены несколько глобальных переменных, сохраняющих коллекцию объектов и важную информацию о состоянии. Также в нее включена функция **getElementStyle()** из рецепта 11.12, необходимая для работы других функций.

Конструктор scrollBar(), формирующий объект с информацией об области прокрутки, получает четыре строковых параметра: идентификатор внешнего контейнера, внутреннего контейнера и идентификаторы двух кнопок прокрутки. Задача конструктора — выполнить некоторую однократную инициализацию и впоследствии упростить выполнение прокрутки. Чтобы обработчики событий у кнопок могли определить, какому набору элементов они соответствуют, в свойство index обеих кнопок помещается значение, соответствующее положению объекта в массиве **scrollBars**. Каждому вновь добавляемому элементу этого массива соответствует индекс, равный значению scrollBars.length.

У каждого создаваемого объекта есть метод initScroller(), вызываемый сразу после того, как объект создан (это происходит в функции initScrollers(), запускаемой при загрузке). Данный метод назначает обработчики событий управляющим кнопкам, а также блокирует контекстное меню в Macintosh (вызываемое, если удерживать кнопку мыши нажатой некоторое время).

Далее следует группа функций, ответственных за обработку событий. Единственная задача handleScr6IIStop() — сбросить флаг, который другие функции используют как признак, разрешающий автоматическую прокрутку, пока функция blockEvent() предотвращает обычную обработку событий **oncontextmenu**. В центре этого приложения лежит функция **handleScrollClick()**, выполняющая прокрутку

в обоих направлениях. Прокрутка производится построчно, а в качестве приближенного размера строки используется высота шрифта содержимого. Каждый элемент img, играющий роль кнопки, имеет свойство index, хранящее положение объекта с информацией в массиве. Дальнейшая идентификация кнопок производится по имени класса, которое определяет изображение как кнопку прокрутки вверх или вниз. Чтобы прокручивать содержимое вверх, необходимо вычитать величину строки из вертикальной координаты вложенного контейнера. В ответ на событие происходит один немедленный вызов scrollBy() (чтобы обеспечить мгновенную реакцию на нажатие), а затем новые вызовы следуют каждые 100 миллисекунд, пока прокрутка не будет прервана по другим причинам (отпущена кнопка мыши).

Управление положением вложенного контейнера с прокручиваемым содержимым — задача функции scrollBy(). Передаваемые ей параметры состоят из индекса объекта, описывающего прокручиваемую область и величину вертикального смещения в пикселах. Если содержимое прокручено не целиком вверх или вниз. с помощью DHTML API функции shiftBy() оно смещается вверх на указанную величину. Но если достигнут предел прокрутки, таймер автоматической прокрутки останавливается и дальше содержимое не перемещается.

Возможности такого контейнера безграничны. Приведенный в этом рецепте код можно изменить, используя для управления не только изображения, но гиперссылки, карты ссылок или помещенный внутрь div текст. Единственное, что нужно сделать, — назначить обработчики событий и удостовериться, что у элемента задано свойство index, связывающее его с нужным объектом scrollBar (как это делается в конструкторе scrollBar().

В то же время некоторые варианты дизайна пагубны. Несмотря на принятую на некоторых сайтах практику использовать для прокручивания перемещение мыши, автоматическая прокрутка также может помешать пользователю, если содержимое важно, так как хорошая автоматическая прокрутка должна быть плавной, но в этом случае она будет слишком долгой для нетерпеливых посетителей.

Объектно-ориентированный подход применен в этом рецепте не случайно. Центральные, часто используемые процедуры (особенно функция scrollByO, вызываемая по таймеру) используют несколько свойств контейнера. Некоторые из этих свойств можно получить, только применяя функцию getElementStyle(), которая должна выполнить довольно большое количество вычислений Вызывать эту функцию каждый раз довольно расточительно. Значения, которые не меняются, пока существует контейнер (например, размеры), должны определяться только один раз, поэтому сохранять их в свойствах объекта для дальнейшего использования — признак хорошего стиля программирования. Побочным продуктом этой методики является возможность сохранять некоторые дополнительные, однократно вычисляемые значения. Также можно ограничить работу кнопок управления, чтобы они не были активны пока страница не загружена целиком. Тогда преждевременные щелчки на кнопках не вызовут ошибок.

Вместо того чтобы использовать фиксированные таблицы стилей, это приложение можно обобщить. Пример обобщения (иллюстрирующий также усложнение кода, нужное для этого) можно увидеть в рецепте 13.3, который, к тому же, предлагает более полноценную вертикальную полосу прокрутки.

Смотрите также

Более сложная полоса прокрутки демонстрируется в рецепте 13.13. Создание объектов объясняется в рецепте 3.8. Как определить настройки стиля, применяемые к элементу по умолчанию, рассказано в рецепте 11.12.

13.13. Создание полосы прокрутки

NN7 IE5

Задача

Необходимо предоставить пользователям возможность прокручивать содержимое некоторой области, используя полосу прокрутки стандартного вида

Решение

В решении задействовано множество элементов HTML, применяемых как для формирования контейнера, так и для имитации полосы прокрутки. Пример HTML-кода приведен в листинге 13.7 в обсуждении. В качестве базового сценария для создания и управления имитированными полосами прокрутки можно использовать библиотеку scrollBars.js, приведенную в листинге 13.8.

К странице нужно подключить две библиотеки: DHTMLAPI.js из рецепта 13.3 и scrollBars.js, а также инициализировать их. Инициализацию следует производить из обработчика события onload тела документа:

```
<body onload="initDHTMLAPI(); initScrollbars()">
```

Вызываемая из initScrollBars() функция не обязательно является частью библиотеки scrollBars.js, так как в ней должна быть информация о каждой полосе прокрутки на странице. Например, в показанной ниже функции initScrollBars() создается один объект, связанный с определенным набором элементов, а также формируется HTML-код полосы прокрутки:

```
function initScrollbars() {
    scrollBars[0] - new scrollBar("pseudoWindow", "outerWrapper",
       "innerWrapper");
    scrollBars[0].appendScroll();
}

<body onload="initDHTMLAPI(); initScrollbars()">
```

Обсуждение

Предлагаемое решение представляет собой расширение рецепта 13.12, включающее гораздо более сложные возможности, например перетаскивание индикатора положения полосы прокрутки и его синхронизация с прокруткой содержимого. Кроме того, компоненты полос прокрутки (изображения и элемент div) создаются динамически, поэтому их не нужно точно позиционировать, их положение

будет зависеть от размеров и стиля контейнера. Результат применения этого решения показан на рис. 13.4.

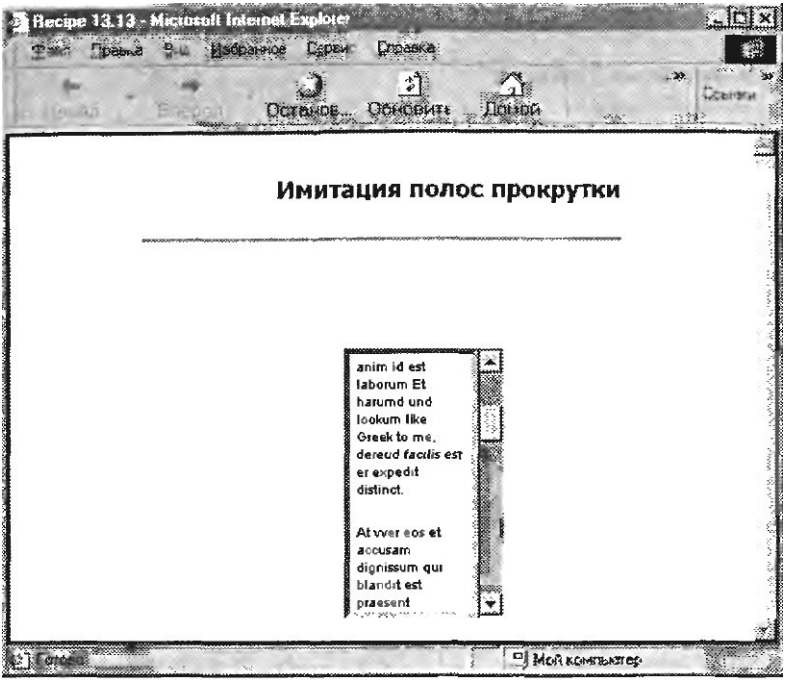


Рис. 13.4. Имитация полос прокрутки для элемента div

В этом решении важную роль играют два предыдущих рецепта. Во-первых, используется библиотека DHTML API из рецепта 13.3. Во-вторых, необходим код для перетаскивания элемента из рецепта 13.11. Одна из функций, **dragIt()**, приспособлена под нужды данного приложения, поэтому все функции перетаскивания помещены в библиотеку scrollBars.js.

Основу решения составляет HTML-код прокручиваемого контейнера и его содержимого, показанный на листинге 13.7. Для самих полос прокрутки HTML-код не нужен, так как он будет автоматически сгенерирован позже.

Листинг 13.7. HTML-код прокручиваемой области

```
<div id="pseudoWindow0" style="position:absolute; top:350px; left:400px">
<div id="outerWrapper0" style="position:absolute; top:0px; left:0px;
    height:150px; width:100px; overflow:hidden; border-top:4px solid
    #666666; border-left:4px solid #666666; border-right:4px solid #cccccc;
    border-bottom:4px solid #cccccc; background-color:#ffffff">
<div id="innerWrapper0" style="position:absolute; top:0px; left:0px;
    padding:5px; font:10px Ventana, Ariel, Helvetica, sans-serif">
<p style="margin-top:0em">Lorem ipsum dolor sit amet, consectetaur...</p>

</div>
</div>
</div>
```

Основной код этого рецепта находится в библиотеке scrollBars.js, код которой показан в листинге 13.8. Он разделен на четыре секции: создание полос прокрутки, функции обработки событий, отслеживание положения и перетаскивание элементов (для индикатора положения).

Листинг 13.8. Библиотека scrollBars.js

```
    Создание полос прокрутки

// Глобальные переменные
var scrollEngaged = false;
var scrollInterval;
var scrollBars = new Array();

// Утилита для определения эффективного значения стиля
function getElementStyle(elemID, IEStyleAttr, CSSStyleAttr) {
    var elem = document.getElementById(elemID);
    if (elem.currentStyle) {
        return elem.currentStyle[IEStyleAttr];
    } else if (window.getComputedStyle) {
        var compStyle = window.getComputedStyle(elem, "" );
        return compStyle.getPropertyValue(CSSStyleAttr);
    }
    return "";


// Конструктор полосы прокрутки
function scrollBar(rootID, ownerID, ownerContentID) {
    this.rootID = rootID;
    this.ownerID = ownerID;
    this.ownerContentID = ownerContentID;
    this.index = scrollBars.length;

    // однократные вычисления, используемые для других манипуляций
    // с полосами прокрутки
    this.rootElem = document.getElementById(rootID);
    this.ownerElem = document.getElementById(ownerID);
    this.contentElem = document.getElementById(ownerContentID);
    this.ownerHeight = parseInt(getElementStyle(ownerID, "height",
       "height"));
    this.ownerWidth = parseInt(getElementStyle(ownerID, "width", "width"));
    this.ownerBorder = parseInt(getElementStyle(ownerID, "borderTopWidth",
        "border-top-width")) * 2;
    this.contentHeight = Math.abs(parseInt(this.contentElem.style.top));
    this.contentWidth = this.contentElem.offsetWidth;
    this.contentFontSize = parseInt(getElementStyle(this.ownerContentID,
        "fontSize", "font-size"));
    this.contentScrollHeight = this.contentElem.scrollHeight;

    // создание объекта quirks, свойства (CSS-совместимые) которого по
    // умолчанию равны 0; относящиеся к делу значения будут записаны в них
    // позже.
```

```
    this.quirks = {on:false, ownerBorder:0, scrollBorder:0,
       contentPadding:0};
    if (navigator.appName == "Microsoft Internet Explorer" &&
        navigator.userAgent.indexOf("Win") != -1 &&
        (typeof document.compatMode == "undefined" ||
        document.compatMode == "BackCompat")) {
        this.quirks.on = true;
        this.quirks.ownerBorder = this.ownerBorder;
        this.quirks.contentPadding =parseInt(getElementStyle(ownerContentID,
        "padding"  "padding"))
     }
    // определяются при инициализации полосы прокрутки
    this.scrollWrapper = null.
    this.upButton = null;
    this.dnButton = null:
    this.thumb = null;
    this.buttonLength = 0:
    this.thumbLength = 0:
    this.scrollWrapperLength = 0
    this.dragZone = {left:0, top:0, right:0, bottom:0]

    // формирование кода полосы прокрутки
    this.appendScroll = appendScrollBar;


// Создание элементов полосы прокрутки и присоединение их к псевдоокну
function appendScrollBar() {
    // размеры кнопок и индикатора положения (настраиваемые)
    var imgH = 16;
    var imgW = 16;
    var thumbH = 27:

    // стрелка "вверх", которая сначала нужна, чтобы задать
    // размер scrollWrapper
    var lineup = document.createElement("img");
    lineup.id = "lineup" + (scrollBars.length  1):
    lineup.className = "lineup":
    lineup.index = this.index;
    lineup.src = "scrollUp.gif"
    lineup.height = imgH;
    lineup.width = imgW:
    lineup.alt = "Scroll Up":
    lineup.style.position = "absolute":
    lineup.style.top = "0px":
    lineup.style.left = "0px".

    // scrollWrapper определяет цвета прокручиваемой "страницы" и 3D-рамки
    var wrapper = document.createElement("div");
    wrapper.id = "scrollWrapper" + (scrollBars.length - 1).
    wrapper.className = "scrollWrapper":
    wrapper.index = this.index,
    wrapper.style.position = "absolute":
```

продолжение

Листинг 13.8 (продолжение)

```
    wrapper.style.visibility = "hidden";
    wrapper.style.top = "0px";
    wrapper.style.left = this.ownerWidth + this.ownerBorder -
        this.quirks.ownerBorder + "px";
    wrapper.style.borderTop = "2px solid #666666";
    wrapper.style.borderLeft = "2px solid #666666";
    wrapper.style.borderRight= "2px solid #cccccc";
    wrapper.style.borderBottom= "2px solid #cccccc";
    wrapper.style.backgroundColor = "#999999";
    if (this.quirks.on) {
        this.quirks.scrollBorder = 2;
    }
    wrapper.style.width = lineup.width + (this.quirks.scrollBorder*2)+"px";
    wrapper.style.height = this.ownerHeight + (this.ownerBorder - 4) -
        (this.quirks.scrollBorder * 2) + "px";

    // стрелка "вниз"
    var linedn = document.createElement("img");
    linedn.id = "linedown" + (scrollBars.length - 1);
    linedn.className = "linedown";
    linedn.index = this.index;
    linedn.src = "scrollDn.gif";
    linedn.height = imgH;
    linedn.width = imgW;
    linedn.alt = "Scroll Down";
    linedn.style.position = "absolute";
    linedn.style.top = parseInt(this.ownerHeight) + (this.ownerBorder - 4) -
        (this.quirks.ownerBorder) - linedn.height + "px";
    linedn.style.left = "0px";

    // Перетаскиваемый индикатор положения фиксированного размера
    var thumb = document.createElement("img");
    thumb.id = "thumb" + (scrollBars.length - 1);
    thumb.index = this.index;
    thumb.src = "thumb.gif";
    thumb.height = thumbH;
    thumb.width = imgW;
    thumb.alt = "Scroll dragger";
    thumb.style.position = "absolute";
    thumb.style.top = lineup.height + "px";
    thumb.style.width = imgW + "px";
    thumb.style.height = thumbH + "px";
    thumb.style.left = "0px";

    // Заполнение свойств объекта scroll Bar значениями свойств
    // отображенных элементов
    this.upButton = wrapper.appendChild( lineup);
    this.thumb = wrapper.appendChild(thumb);
    this.dnButton = wrapper.appendChild(linedn);
    this.scrollWrapper = this.rootElem.appendChild(wrapper);
    this.buttonLength = imgH;
    this.thumbLength = thumbH;
```

```
    this.scrollWrapperLength=parseInt(getElementStyle(this.scrollWrapper id.
        "height"  "height")):
    this.dragZone.left = 0;
    this.dragZone.top = this.buttonLength:
    this.dragZone.right = this.buttonLength:
    this.dragZone.bottom = this.scrollWrapperLength - this buttonLength -
        (this.quirks.scrollBorder * 2)

    // все события передаются элементу scrollWrapper
    this.scrollWrapper.onmousedown = ha dleScrollClick:
    this.scrollWrapper.onmouseup = handleScrollStop:
    this.scrollWrapper.oncontextmenu = blockEvent:
    this.scrollWrapper.ondrag = blockEvent;

    // Теперь можно показывать
    this.scrollWrapper.style.visibility = "visible".
}

/*********************************
    Функции обработки событий
*********************************/
// обработчик onmouseup
function handleScrollStop0 {
    scroll Engaged = false:
}

// Предотвращение появления контекстного меню на Mac при удержании нажатой
// кнопки
function blockEvent(evt) {
    evt = (evt) ? evt : event;
    evt.cancel Bubble = true;
    return false:
}

// обработчик click
function handleScrollClick(evt) {
    var fontSize, contentHeight:
    evt - (evt) ? evt . event:
    var target - (evt.target) ? evt.target : evt.srcElement;
    target - (target.nodeType — 3) ? target.parentNode : target:
    var index - target.index;
    fontSize - scrollBars[index].contentFontSize;
    switch (target.className) {
        case "lineup" :
            scroll Engaged = true:
            scrollBy(index, parseInt(fontSize));
            scrollInterval » setInterval("scrollBy(" + index + ". " +
                parseInt(fontSize) + ")". 100):
            evt.cancel Bubble = true:
            return false:
            break;
        case "linedown" .
            scrollEngaged = true:
```

продолжение

Листинг 13.8 (продолжение)

```
            scrollBy(index, -(parseInt(fontSize)));
            scrollInterval = setInterval("scrollBy(" + index + "     +
                parseInt(fontSize) " ) " , 100);
            evt.cancelBubble = true;
            return false;
            break;
        case "scrollWrapper"
            scrollEngaged = true;
            var evtY = (evt.offsetY) ? evt.offsetY   ((evt.layerY) ?
              evt.layerY • -1);
            if (evtY >= 0) {
                var pageSize = scrollBars[index].ownerHeight - fontSize;
                var thumbElemStyle = scrollBars[index].thumb.style;
              // устанавливаем отрицательное значение, чтобы двигать
              // документ вверх
                if (evtY > (parseInt(thumbElemStyle.top) +
                    scrollBars[index].thumbl_ength)) {
                    pageSize = -pageSize;
                }
                scrollBy(index, pageSize);
                scrollInterval = setInterval("scrollBy(" + index + "   " +
                    pageSize + " ) " , 100);
                evt.cancelBubble = true;
                return false;
            }
    }
    return false;
}

// Активизация прокрутки содержимого
function scrollBy(index, px) {
    var scroller = scrollBars[index];
    var elem = document.getElementById(scroller.ownerContentID),
    var top = parseInt(elem.style.top);
    var scrollHeight = parseInt(elem.scrollHeight);
    var height = scroller.ownerHeight;
    if (scrollEngaged && top+px >= -scrollHeight+height && top + px <= 0) {
        shiftBy(elem, 0, px);
        updateThumb(index);
    } else if (top + px < -scrollHeight + height) {
        shiftTo(elem,0,-scrollHeight+height-scroller.quirksContentPadding);
        updateThumb(index);
        clearInterval(scrollInterval);
    ] else if (top + px > 0) {
        shiftTo(elem, 0, 0);
        updateThumb(index);
        clearInterval(scrollInterval);
    } else {
        clearInterval(scrollInterval);
    }
}
```

```
    Отслеживание положения

// Позиционирование индикатора положения после прокрутки
function updateThumb(index) {
    var scroll = scrollBars[index];
    var barLength=scroll.scrollWrapperLength-(scroll.quirks.scrollBorder*2).
    var buttonLength = scroll.buttonLength;
    barLength -= buttonLength * 2;
    var docElem = scroll.contentElem;
    var docTop = Math.abs(parseInt(docElem.style.top));
    var scrollFactor = docTop/(scroll.contentScrollHeight
       scroll.ownerHeight);
    shiftTo(scroll.thumb 0. Math.round((barLength - scroll.thumbLength) *
        scrollFactor) + buttonLength);


// Позиционирование содержимого по положению индикатора прокрутки
function updateScroll() {
    var index = selectedObj.index;
    var scroller = scrollBars[index];

    var barLength = scroller.scrollWrapperLength -
       (scroller.quirks.scrollBorder * 2).
    var buttonLength = scroller.buttonLength;
    var thumbLength = scroller.thumbLength;
    var wellTop = buttonLength;
    var wellBottom = barLength - buttonLength - thumbLength;
    var wellSize = wellBottom - wellTop;
    var thumbTop = parseInt(getElementStyle(scroller.thumb.id, "top",
       "top"));
    var scrollFactor = (thumbTop - buttonLength)/wellSize;
    var docElem = scroller.contentElem;
    var docTop = Math.abs(parseInt(docElem.style.top));
    var scrollHeight = scroller.contentScrollHeight;
    var height = scroller.ownerHeight;
    shiftTo(scroller.ownerContentID, 0.-(Math.round((scrollHeight - height)*
        scrollFactor)));
}

   Перетаскивание элементов
*********************************/
// Глобальная переменная со салкой на выделенный элемент
var selectedDbj;
// Координата указателя мыши относительно элемента
var offsetX, offsetY;
var zone = {left:0, top:16, right:16, bottom:88};

// Установка глобальной ссылки на перетаскиваемый элемент
function setSelectedElem(evt) {
    var target = (evt.target) ? evt.target : evt.srcElement;
    target = (target.nodeType && target.nodeType == 3) ? target.parentNode :
       target;
```

продолжение ☞

Листинг 13.8 (продолжение)

```
    var divID = (target.id.indexOf("thumb") != -1) ? target.id  "";
    if (divID) {
        if (document.layers) {
            selectedObj = document.layers[divID];
        ] else if (document.all) {
            selectedObj = document.all(divID);
        } else if (document.getElementById) {
            selectedObj = document.getElementById(divID);
        }
        setZIndex(selectedObj, 100):
        return:
    }
    selectedObj - null;
    return:
}
// Перетаскивать индикатор можно только в пределах полосы прокрутки
function dragIt(evt) {
    evt - (evt) ? evt : event:
    var x. y. width, height:
    if (selectedObj) {
        if (evt.pageX) {
            x - evt.pageX - offsetX:
            y - evt.pageY - offsetY;
        } else if (evt.clientX || evt.clientY) {
            x - evt.clientX - offsetX;
            y - evt.clientY - offsetY:
        }
        var index - selectedObj.index;
        var scroller - scrollBars[index];
        var zone = scroller.dragZone;
        width = scroller.thumb.width;
        height = scroller.thumb.height;
        x = (x < zone.left) ? zone.left : ((x + width > zone.right) ?
           zone.right - width : x);
        y = (y < zone.top) ? zone.top : ((y + height > zone.bottom) ?
           zone.bottom - height : y):
        shiftTo(selectedObj, x, y):
        updateScroll();
        evt.cancel Bubble - true:
        return false:
    }
}

// "Включаем" выделенный элемент и устанавливаем смещение относительно
// указателя
function engage(evt) {
    evt = (evt) ? evt : event:
    setSelectedElem(evt);
    if (selectedObj) {
        if (document.body && document.body.setCapture) {
           // активизация механизма перехвата событий в IE/Win
            document.body.setCapture();
        }
```

```
        if (evt.pageX) {
            offsetX = evt.pageX - ((typeof selectedObj.offsetLeft !=
              "undefined") ?
                      selectedObj.offsetLeft : selectedObj.left);
            offsetY = evt.pageY - ((selectedObj.offsetTop) ?
                      selectedObj.offsetTop : selectedObj.top);
        } else if (typeof evt.clientX != 'undefined') {
            offsetX = evt.clientX - ((selectedObj.offsetLeft) ?
                      selectedObj.offsetLeft : 0);
            offsetY = evt.clientY - ((selectedObj.offsetTop) ?
                      selectedObj.offsetTop : 0);
        }
        return false;
    }
}

// "Выключаем" выделенный элемент
function release(evt) {
    if (selectedObj) {
        setZIndex(selectedObj, 0);
        if (document.body && document.body.releaseCapture) {
           // прекращаем перехват событий в IE/Win
            document.body.releaseCapture();
        }
        selectedObj = null;
    }


// назначаем обработчики событий, используемые как в IE, так и в Netscape
function initDrag() {
    if (document.layers) {
       // активизация перехвата событий в модели событий NN4
        document.captureEvents(Event.MOUSEDOWN | Event.MOUSEMOVE |
          Event.MOUSEUP);
        return;
    } else if (document.body && document.body.addEventListener) {
       // активизация перехвата нужных событий в модели W3C DOM
       document.addEventListener("mousedown", engage, true);
       document.addEventListener("mousemove", dragIt, true);
       document.addEventListener("mouseup", release, true);
       return;
    }
   document.onmousedown = engage;
   document.onmousemove = dragIt;
   document.onmouseup = release;
   return;
}
```

Код библиотеки начинается с определения ряда глобальных переменных. Далее идет вспомогательная функция из рецепта 11.12, getElementStyle(), используемая позже при создании полос прокрутки.

Функция-конструктор scrollBar() требует передачи трех строковых аргументов: идентификатора div, в котором расположено содержимое и в который будет помещена полоса прокрутки (далее этот div неформально называется корневым

контейнером), идентификатора внешнего контейнера с прокручиваемым содержимым и идентификатора внутреннего контейнера. Задача конструктора — выполнить некоторые необходимые позже однократные вычисления и инициализацию, нужные каждой полосе прокрутки в отдельности (на одной странице допустимо использовать несколько полос прокрутки). Чтобы функции обработки событий могли понять, с какой из полос прокрутки работать, в свойство index обеих кнопок прокрутки помещено значение, равное числовому индексу в массиве scrollBars. При создании каждой новой полосы прокрутки ей соответствует значение индекса, равное scrollBars.length, так как оно на единицу больше максимального значения индекса в массиве.

Многочисленные свойства каждого из объектов scrollBar не получают реальных значений до тех пор, пока полоса прокрутки не будет физически создана на странице. В конструкторе также имеется секция кода, предназначенная для браузеров, работающих в «капризном» режиме (то есть не CSS-совместимом), таких как IE 5 и 5.5. Причина в том, что на размеры элемента влияют такие факторы, как размеры рамок и величина заполнения, интерпретирующиеся по-разному в «капризном» и CSS-совместимом режиме. Одно из свойств, dragZone, используется, чтобы задать область, в пределах которой можно перетаскивать индикатор положения.

Следующая функция, appendScrollBar(), — настоящий монстр. Она легко может быть разбита на несколько частей, но структура достаточно проста, чтобы сохранить ее целиком, так как функция формирует составляющие полосу прокрутки объекты DOM. По мере того как становятся доступными отдельные свойства, их значения записываются в соответствующие поля абстрактного объекта scrollBar, созданного конструктором. Полоса прокрутки состоит из одного контейнера (который также служит серым фоном) и трех изображений: двух кнопок и перетаскиваемого индикатора положения. Обработчики событий мыши назначаются контейнеру, чтобы получать все сообщения от любых компонентов полосы. Изначально полоса прокрутки создается невидимой и отображается только в конце, чтобы обойти ошибку отображения в IE/Windows, из-за которой полоса позиционируется неправильно.

Следующая секция кода состоит из функций обработки событий. Единственная задача handleScrollStop() — сбрасывать флаг, который разрешает автоматическую прокрутку (при удержании нажатой кнопки прокрутки), при этом blockEvents() блокирует обработку события oncontextmenu на Macintosh. Обработка щелчков мыши на областях прокрутки на одну страницу вверх и вниз производится функцией handleScrollClick(). В ней есть три ветви, вычисляющие расстояние, на которое прокручивается содержимое. Самая тонкая часть этой функции — определение, был ли щелчок произведен в верхней или нижней области, это нужно, чтобы задать направление прокрутки.

По сравнению с функцией scrollBy() из рецепта 13.12, в этом рецепте та же функция должна выполнять более сложные действия. В ответ на щелчок пользователя на кнопке не только должно прокручиваться содержимое, но и в соответствии с текущим положением должно изменяться положение индикатора прокрутки. Поэтому после каждого вызова DHTML API функций shiftBy() и shiftTo() вызывается функция updateThumb(). Обратите внимание, что в коде scrollBy() есть

две дополнительные ветви. Они учитывают ситуации при постраничной прокрутке, когда осталось менее одной страницы. В этих случаях производится прокрутка (и позиционирование индикатора) до самого верха или низа.

Третья секция кода служит для синхронизации положения содержимого и индикатора прокрутки. Для этого функция updateThumb() определяет величину прокрутки документа и в соответствии с ней позиционирует индикатор. Функция updateScroll(), напротив, приводит прокрутку документа в соответствие с положением индикатора, когда пользователь перемещает его. Пропорция между положением индикатора между кнопками прокрутки и величиной прокрутки страницы известна (после некоторых вычислений, в которых участвует текущий размер полосы и составляющих ее изображений). Благодаря этой функции документ в реальном времени продвигается вверх или вниз по мере того, как пользователь перемещает индикатор.

Последняя секция содержит код, необходимый для перетаскивания индикатора. Основы перетаскивания элементов из рецепта 13.11 применены здесь для перетаскивания элемента полосы прокрутки. Хотя имена используемых функций не изменены (engage(), dragIt() и release()), ряд моментов требуется изменить, чтобы соответствовать специфике полосы прокрутки. Немного изменена функция setSelectedElement() (вызываемая из engage()), чтобы реагировать на элементы, идентификатор которых содержит строку thumb. Кроме того, хотя полосы прокрутки не рассчитаны на работу в браузерах четвертой версии, код для поддержки перетаскивания в них оставлен на месте.

Наибольшим изменениям подверглась функция dragIt(). Изменения показаны в листинге 13.8 жирным шрифтом и предназначены, прежде всего, для того, чтобы ограничить область допустимых перемещений индикатора вдоль полосы прокрутки, удерживая его между кнопками. Значения, управляющие границами, хранятся в глобальной переменной zone. Кроме того, модифицированная функция dragIt() вызывает updateScroll(), чтобы синхронизировать прокрутку с новым положением индикатора.

Чтобы создать полосу прокрутки и подготовить ее к взаимодействию с пользователем, применяются две ключевые функции: конструктор scrollBarQ, получающий идентификаторы элементов HTML, показанных в начале этого решения, и appendScroll(). Вызывать эти две подпрограммы следует из обработчика события onload страницы после инициализации DHTML API.

Дизайн предлагаемых полос прокрутки, показанный на рис. 13.4, очень традиционен. Но это не значит, что вы каким-то образом ограничены этим стилем. Например, вы можете убрать кнопки, оставив только стилизованный индикатор прокрутки, или имитировать обычные для операционный системы пользователя полосы прокрутки, разработав отдельные рисунки для каждой из платформ. Стандартные полосы прокрутки выглядят по-разному в Windows 9x, Windows XP, Mac OS 9 и Mac OS X.

Следует считаться еще с одним фактором: в Mac OS 9 и старше кнопки прокрутки расположены не сверху и снизу, а вместе в нижней части полосы прокрутки. Не то чтобы пользователи Mac могут не понять, как пользоваться разнесенными кнопками, но такой стиль может показаться неестественным для тех, кто привык к новому интерфейсу. Код этого решения можно модифицировать,

чтобы кнопки прокрутки располагались вместе. Эти изменения затронут многие вещи, в частности позиционирование индикатора, но можно сделать в коде ответвления (или подгружать разные версии библиотек для разных ОС).

Горизонтальная прокрутка в этом рецепте не рассматривается. Если все же она нужна, можно сделать ряд изменений в коде. Прежде всего следует изменить все вызовы функций scrollBy(), shiftBy() и shiftTo(), в которых необходимо поменять порядок параметров, чтобы координата y была равна нулю, а координата x менялась. При определении размеров элементов требуется рассматривать не высоту, а ширину. К счастью, все методы работы в нестандартном режиме прямо переносятся на горизонтальные полосы, поэтому вам не придется перелопачивать эти части кода.

Полосы прокрутки — одно из наиболее объемных приложений в этой книге, но оно построено на основе других рецептов, без изменения их инфраструктуры (особенно это касается DHTML API и перетаскивания элементов). Этот рецепт также демонстрирует возможность реализовать большую часть пользовательского интерфейса (по крайней мере, в современных браузерах), даже в противоположность модели обычного печатного документа.

Смотрите также

Необходимая для рецепта библиотека DHTML API показана в рецепте 13.3. Процедуры для перетаскивания элементов приведены в рецепте 13.11. Прокрутка только с помощью кнопок демонстрируется в рецепте 13.12. Дополнительную информацию о вспомогательной функции getElementStyle() можно найти в рецепте 11.12.

14 Динамическое содержимое

14.0. Вступление

Браузеры с JavaScript всегда позволяли определенным образом контролировать содержимое, давая возможность влиять на то, что посетитель видит на странице. Но сложная объектная модель документа и возможности автоматического заполнения страниц с таких браузерах, как Internet Explorer 4 и Netscape 6, дали программистам карт-бланш в управлении содержимым как при загрузке страницы, так и после нее (конечно, в рамках ограничений безопасности). Эта глава посвящена тому, как генерировать содержимое страницы, а также как управлять существующим. В следующей главе показано несколько специфических применений этих возможностей.

Программисты, уделяющие большую часть времени разработке серверных программ, часто недооценивают возможности, которые могут дать скучной и мертвой странице клиентские сценарии. Ход их мыслей (довольно логичный) таков: заставить сервер выполнять всю работу, формировать содержимое и отправлять его своим путем в браузер, где пользователь читает переданную информацию и, возможно, вводит что-то в формы. Затем браузер посылает ответ на сервер, где еще одна программа обрабатывает пользовательские данные. Это мощный серверный механизм, необходимый для приложений, использующих транзакции и базы данных.

Тем не менее пользователям привычнее прямая манипуляция данными и мгновенная реакция на действия, как в самостоятельных приложениях, работающих на локальном компьютере. Когда в текстовом процессоре изменяются настройки шрифта, изменения видны сразу; когда сортируются ячейки таблицы, сортировка происходит мгновенно. Ожидать же, пока завершится отсылка формы на сервер, обработка ее и доставка перестроенной страницы, не так уж приятно, даже если имеется высокоскоростное подключение к Интернету.

За исключением тех ситуаций, когда данные в базе на сервере меняются так часто, что требуется постоянное обновление их у клиента, некоторый набор данных можно передать с документом, а различные преобразования вида этих данных производить целиком на стороне клиента. Это не только обеспечит мгновенную реакцию приложения, но и разгрузит сервер.

Эта и следующая главы посвящены страницам, которые могут выглядеть совершенно по-разному, потому что пользователь может сортировать в них таблицы, менять текст документа и даже фильтровать содержимое на основе своих настроек. Право выбора предоставляется пользователям, в известном смысле.

Только время покажет, будут ли серверные программисты чувствовать себя достаточно комфортно в программировании клиентских приложений. Страшные истории о несовместимости между браузерами различных версий и фирм все меньше отвечают действительности по мере роста числа W3C DOM совместимых браузеров.

14.1. Формирование содержимого при загрузке страницы

NN2 **IE3**

Задача

Необходимо менять содержимое страницы в зависимости от пользовательских настроек или cookie, желательно обратно-совместимым способом.

Решение

Во всех поддерживающих сценарии браузерах в любом месте страницы, где должно быть изменяемое содержимое, можно внедрять команду document.write(). Следующий код отображает на странице текст, соответствующий операционной системе пользователя:

```
<html>
<head>
<script  type="text/javascript">
function yourOS() {
    var ua = navigator.userAgent.toLowerCaseO:
    if (ua.indexOf("win") != -1) {
        return "Windows":
    ] else if (ua.indexDf("mac") != -1) {
        return "Macintosh":
    } else if (ua.indexOf("linux") != -1) {
        return "Linux":
    } else if (ua.indexOf("x11") !=  1) {
        return "Unix"
    ] else {
        return "Computers";
    }
}
</script>
<body>
<h1>Welcome to GiantCo Computers</h2>
```

```
<h2>We love
<script type="text/javascript">document.write(yourOS())</script>
<noscript>Computers</noscript>
Users!</h2>

</body>
</html>
```

Обсуждение

Предыдущее решение работает во всех поддерживающих сценарии браузерах. Тем не менее экспериментировать с document.write() следует с осторожностью. Когда эта команда внедряется внутрь страницы (как это и делается в решении), использовать обычно ассоциируемые с document.write() методы document.close() и document.open() не нужно. Причина в том, что в процессе загрузки поток отображения страницы уже открыт, закрывает его браузер также автоматически по окончании содержимого.

Частая ошибка начинающих — попытка использовать document.write() для модификации уже загруженной страницы. Если попытаться так сделать, все содержимое страницы автоматически стирается, унося сценарии и встроенные в страницу данные. Простой вызов document.write() эквивалентен трем последовательным вызовам: document.clear(), document.open() и document.write(). Другими словами, после того как страница загружена, применять document.write() нужно только для того, чтобы целиком заменить страницу сгенерированным содержимым (см. рецепт 14.12). Чтобы изменить текущую страницу (в той мере, в какой эта возможность поддерживается целевым браузером), необходимо применять более непосредственные методы работы с элементами, описываемые в главе 15.

Смотрите также

В рецепте 15.1 показано, как использовать document.write() для отображения на странице случайной фразы. Как поприветствовать пользователя в соответствии с его временем суток, рассказано в рецепте 15.6.

14.2. Динамическое формирование нового содержимого

NN2 **IE3**

Задача

С помощью сценария необходимо сформировать новое содержимое, которое должно полностью заменить старую страницу.

Решение

Приведенный ниже пример кода получает из формы введенный пользователем текст и формирует на его основе полностью новую страницу, замещающую старую:

```
<html>
<head>
<title>Welcome Page</title>
<script type="text/javascript">
// Создаем новую страницу и заменяем ей текущий документ
function rewritePage(form) {
    // сборка HTML-кода новой страницы
    var newPage = "<html><head><title>Страничка для ";
    newPage += form.entry.value;
    newPage += "</title></head><body bgcolor='#ffffcc'>";
    newPage += "<n1>Привет. " + form.entry.value + "!</h1>";
    newPage += "</body></html>";
    // запись кода за один раз
    document.write(newPage);
    // закрытие потока записи
    document.close();
}
</script>
<body>
<h1>Добро пожаловать!</h1>
<hr>
<form onsubmit="return false:">
<p>Введите ваше имя: <input type="text" name="entry" id="entry"></p>
<input type="button" value="Новая страница" onclick="rewritePage(this.form);">
</form>
</body>
</html>
```

Обсуждение

Так как единственный вызов метода document.write() в уже загруженной странице полностью заменяет ее, необходимо собрать весь код новой страницы в одну строковую переменную, а затем вызвать document.write(), передав в качестве единственного параметра собранную строку. Помните, что нужно сформировать код всей страницы целиком, поэтому в строку должна попасть и головная секция кода (например, описание названия документа). Если не указать теги элементов html, head или body, браузер будет интерпретировать переданный текст как содержимое тела документа и допишет недостающие теги сам. Но лучше все же указывать эти теги самостоятельно.

Строка, задающая новую страницу, может содержать любой корректный HTML-код, в том числе включающий сценарии. Таким образом, можно передавать в новую страницу значения переменных, вписывая их в код сценария новой страницы. Например, если нужно передать значение строковой переменной, названной myName, для формирования кода можно применить следующее выра-

жение. Обратите внимание на то, что вокруг строкового значения расставляются кавычки, как в обычном сценарии:

```
htmlString - "var myName = '" + myName + "':";
```

Если попытаться применить document.write() для формирования тегов **script** в некоторых устаревших браузерах, можно столкнуться с неожиданной проблемой, так как закрывающий тег **</script>**, входящий в строку, браузер может воспринять как завершение формирующего страницу сценария. Чтобы избежать появления этой проблемы, следует разбить закрывающий тег на две части и заменить символ косой черты escape-последовательностью, как показано ниже:

```
htmlString += 'script type="text/javascript">текст сценария<\/scr' + 'ipt>';
```

Вызов метода document.write() важно завершать вызовом document.close(), который выполняется, несмотря на то что предыдущая команда уничтожает и страницу и сценарий. Если не включать вызов document.close(), на новой странице может отобразиться не все содержимое, в особенности это касается случаев, когда в новой странице есть изображения.

Смотрите также

Как собрать строку из отдельных фрагментов, показано в рецепте 1.1.

14.3. Внедрение внешнего HTML

NN6 IE5

Задача

В документ на странице нужно внедрить HTML-код из другого HTML-документа.

Решение

Можно поместить внешний документ в элемент **iframe** и замаскировать его так, чтобы он выглядел обычной частью страницы. Вот пример описания **iframe**, который незаметно сливается с обычной HTML-страницей:

```
<iframe id="myFrame" frameborder="0" vspace="0" hspace="0" marginwidth="0"
marginheight="0" width="100%" src="external.html" scrolling="no"
style="overflow:visible"></iframe>
```

Чтобы корректно задать размер **iframe**, следует подождать, пока он загрузит свой документ, а затем определить высоту содержимого. Ниже приведен пример функции, вызываемой из обработчика события onload на странице и подстраивающей высоту элемента **iframe**, идентификатор которого ей передан:

```
function adjustIFrameSize(id) {
    var myIframe = document.getElementById(id);
    if (myIframe) {
```

```
if(my Iframe . contentDocument &&
   myIframe.contentDocument.body.offsetHeight){
   // Синтаксис W3C DOM (и Mozilla)
    myIframe.height = myIframe contentDocument.body.offsetHeight;
}else if (myIframe.Document && myIframe.Document.body.scrollHeight){
   // Синтаксис IE DOM
    myIframe.height = myIframe.Document.body scrollHeight;
}
```

Для приведенного в примере элемента вызов функции должен выглядеть так:

```
onload = "adjustIFrameSize('myFrame');"
```

Обсуждение

Чтобы приведенная функция подстройки размера могла работать, оба документа должны быть доставлены с одного сервера и домена, иначе сценарий не сможет обратиться к содержимому **iframe.** Обращение к содержимому выглядит не так просто, как при работе с фреймами. В разных объектных моделях для обращения к содержимому служат разные свойства, в IE — это Document, а в W3C DOM совместимых браузерах — **contentDocument.** Эти свойства содержат ссылку на корень внедренного документа Используя ее, можно получить доступ к элементу body и к другим элементам этого документа. В этом примере высота документа (неизвестная до окончания загрузки) управляет высотой содержащего его **iframe.**

Остерегайтесь использования в **iframe** гиперссылок, так как если не присвоить атрибуту гиперссылки **target** значение **_top,** новая страница будет загружена внутри **iframe.** Кроме того, если новая загруженная в **iframe** страница будет другой высоты, она будет либо усечена (если страница слишком длинная) или же останется пустой интервал (страница слишком маленькая). Единственное, что можно сделать, — изменить размер элемента в соответствии с новым содержимым.

Чтобы заставить **iframe** автоматически подстраивать свой размер, необходимо прежде всего назначить ему обработчик событий onload. При этом в Netscape 6 для этого элемента может не работать обработчик, назначенный через тег или через свойство, хотя применение современных методов назначения обработчиков приводит к желаемому результату. Поэтому при назначении обработчика нужны несколько ветвей, соответствующие разным моделям события. Чтобы приведенная в решении функция могла назначать обработчики, ее следует изменить так:

```
function adjustIFrameSize(id) {
    var mylframe = document.getElementById(id);
    if (myIframe) {
        if (mylframe .contentDocument &&
           my Iframe.contentDocument.body.offsetHeight) {
            // Синтаксис W3C DOM (и Mozilla)
            myIframe.height = myIframe.contentDocument.body.offsetHeight;
        }else if (myIframe.Document && myIframe.Document.body.scrollHeight){
            // Синтаксис IE DOM
            myIframe.height = myIframe.Document.body.scrollHeight;
```

```
        // назначаем элементу iframe обработчик события onload
        if (myIframe.addEventListener) {
            myIframe.addEventListener("load", resizeIframe, false);
        } else {
            myIframe.attachEvent("onload"  resizeIframe);
        }
    }
}
```

Здесь при появлении события onload будет вызываться функция **resizeIframe()**, которая определяет идентификатор получившего событие элемента и использует его для вызова **adjustFrame()**:

```
function resizeIframe(evt) {
    evt = (evt) ? evt : event;
    var target = (evt.target) ? evt.target : evt.srcElement;
    // Учитываем, что в модели W3C событие возникает в документе
    // внутри iframe
    if (target.nodeType == 9) {
      if (evt.currentTarget && evt.currentTarget.tagName.toLowerCase() ==
          "iframe") {
          target = evt.currentTarget;
      }
    }
    if (target) {
        adjustIFrameSize(target.id);
```

Сложность в том, что в W3C DOM браузерах событие onload возникает не в самом элементе, а в содержащемся в нем документе. К счастью, благодаря механизму передачи событий оно достигает iframe, ссылка на который будет помещена в свойство **currentTarget** объекта события. При вызове adjustFrameSize() будут повторно назначены события, но в этом нет ничего страшного. Но никто не мешает вам передвинуть назначение обработчиков в общую функцию инициализации главной страницы в ответ на событие onload (если такая функция есть). При этом обработчики будут назначаться однократно.

Смотрите также

Работа с несовместимыми моделями событий описывается в рецепте 9.1. Подробности об обработчике onload вы можете найти в рецепте 9.2.

14.4. Внедрение данных XML

NN6 **IE5(Win)**

Задача

В работе сценария на странице нужно использовать данные, хранящиеся в документе XML.

Решение

В IE 5 для Windows и Netscape 6 (и в более новых браузерах) имеется возможность создавать виртуальные документы, содержащие данные XML в их исходном виде. При загрузке такого документа следует учитывать браузер, но после загрузки документа для обращения к содержимому и узлам используется платформенно-независимый синтаксис.

Ниже приведена функция **verifySupport()**, загружающая внешний документ XML, URL которого ей передано в глобальную переменную xDoc, и возвращающее булевское значение, сигнализирующее о поддержке браузером нужных возможностей:

```
var xDoc:
// Проверка поддержки браузером XML и загрузка внешнего .xml-файла
function verifySupport(xFile) {
    if (document.implementation && document.implementation.createDocument) {
       // Это способ W3C DOM. поддерживаемый пока что только NN6+
        xDoc - document.implementation.createDocument(""". "theXdoc". null):
    } else if (typeof ActiveXObject !- "undefined') {
       // убеждаемся в действительной поддержке объекта
       // (увы. пользователи IE5/Mac)
        if (document.getElementById("msxml").async) {
            xDoc - new ActiveXObject("Msxml.DOMDocument");
        }
    }
    if (xDoc && typeof xDoc.load !- "undefined") {
       // загрузка внешнего файла (с того же домена)
        xDoc.load(xFile):
        return true:
    } else {
        var reply - confirm("Для этого примера нужен браузер с поддержкой"
       + " XML. такой как IE5+/Windows или Netscape 6+.\n \Вернуться"
       + " назад ?"):
        if (reply) {
            history.back0:


    return false:
}
```

Для работы этой функции необходим вспомогательный элемент, задающийся тегом **<object>** в нижней части страницы и пытающийся загрузить элемент управления ActiveX (в IE), ответственный в этом браузере за работу с XML:

```
<!- Попытка загрузить необходимый для проверки ActiveX
    Msxml.DOMDocument->
<object id="msxml" width="1" height="1"
    classid="CLSID:2933BF90-7B36-11d2-B20E-00C04F983E60" ></object>
```

Обсуждение

В Internet Explorer контейнером для данных XML служит элемент управления ActiveX (MSXML). С последними версиями браузеров поставляются более новые версии этого элемента, но в этом решении используется обладающий боль-

шей обратной совместимостью вариант. С новыми браузерами устаревшие версии этого ActiveX продолжают поставляться.

Хотя W3C DOM Level 2 не предоставляет объектов, которые можно применять для загрузки XML, разработчики Mozilla заполнили этот пробел. Взяв за основу стандартный объект W3C DOM document.implementation и метод createDocument(), в Mozilla к создаваемому createDocument() объекту добавили метод load(). Хотя этот метод можно было бы улучшить, так как распознаваемые им документы XML должны находиться в той же папке, что и вызвавшая его страница. В любом случае, как в IE, так и в Mozilla загружаемые данные должны находиться на том же сервере (и домене), что и сама страница. Таким образом, не стоит рассматривать эту методику как способ собирать данные XML из других источников (хотя с применением серверного программирования эта задача легко решается).

Причина, по которой для загрузки XML приходится использовать специальные виртуальные документы, в том, что если загрузить XML во фрейм (видимый или невидимый), большинство браузеров преобразуют его в HTML. Например, IE преобразует документ, оформляя его содержимое разнообразными цветами, отступами и символами иерархии. Таким образом, структура исходного документа после загрузки будет окончательно погребена созданным браузером оформлением. Многие другие браузеры работают с XML так, как будто это стандартный HTML, вследствие чего в структуру узлов документа попадают теги элементов html, head и body. Если ваши XML-данные содержат теги с такими именами (особенно часто встречаются элементы с именами head и body), обрабатывающий данные сценарий может попасть в бесконечный цикл.

Предложенное решение оформлено в виде функции, возвращающей логическое значение, сигнализирующее о том, что XML поддерживается браузером и данные были успешно загружены. Возвращаемое значение можно использовать как переключатель, определяющий, производить ли действия с загруженным XML-документом или пропустить этот участок кода. Так как IE сообщает об успехе загрузки до того, как документ целиком загружен, вызывать работающую с загруженными данными функцию следует через setTimeout():

```
// Начальная инициализация, вызываемая событием onload
function init(xFile) {
    // Проверяем поддержку XML и загружаем документ .xml
    if (verifySupport(xFile)) {
        // задержка, чтобы процесс загрузки мог догнать поток выполнения
        setTimeout("operativeFunction()" 1000);
    }
}

<body onload="init('myData.xml');">
```

Internet Explorer для Windows поддерживает еще один дополнительный способ включения XML в страницу: островок XML-данных (XML data island). IE позволяет использовать теги <xml>, между которыми можно помещать любой XML-код. Обычно эти данные представляют собой оформленный в виде XML результат запроса к базе данных. Такой формат можно легко использовать для прямого преобразования в динамически генерируемый HTML-код или в данные JavaScript (массивы или объекты), которые затем можно использовать для генерации HTML и выполнения таких операций, как сортировка табличных данных.

Чтобы предотвратить появление на экране встроенных таким образом в страницу данных, следует применить таблицы стилей, присвоив свойству display элемента xml значение none. Ниже приведен оформленный таким образом фрагмент XML-файла, содержащего исторические данные финала Чемпионата мира:

```
<xml id="myData" style="display:none">
<worldcup>
    <final>
        <location>Uruguay</location>
        <year>1930</year>
        <winner>Uruguay</winner>
        <winscore>4</winscore>
        <loser>Argentina</loser>
        <losscore>2</losscore>
    </final>
    <final>
        <location>Italy</location>
        <year>1934</year>
        <winner>Italy</winner>
        <winscore>2</winscore>
        <loser>Czechoslovakia</loser>
        <losscore>1</losscore>
    </final>

</worldcup>
</xml>
```

Сценарии могут использовать для обращения к островкам xml данных, применяя стандартную модель доступа к узлам документа. Например, чтобы получить массив элементов final из предыдущего примера, можно взять такой код:

```
var allCups - document.getElementById("myData").getElementById("final");
```

Смотрите также

Генерация HTML-кода на основе XML-данных описывается в рецепте 14.6. Рецепт 14.8 показывает, как формировать на основе XML объекты JavaScript. Обход дерева узлов XML или HTML-документа демонстрируется в рецепте 14.17.

4.5. Хранение данных в виде объектов JavaScript

NN4 **IE4**

адача

Данные, сформированные серверным процессом, необходимо включить в страницу, так чтобы сценарий на ней мог манипулировать ими и/или предоставлять пользователю варианты их отображения.

Решение

Выполняемый на сервере код может вставлять в готовую к отправке клиенту страницу блоки динамически формируемых данных, и хотя для хранения строковых данных можно использовать скрытые поля форм, структура данных JavaScript обладает гораздо большей гибкостью и мощью.

Идеальной структурой для хранения простых наборов значений являются массивы JavaScript, особенно если применить сокращенный синтаксис описания массива:

```
var dataArray = ["значение0" "значение1" "значение2" "значением"];
```

Для хранения данных, структура которых запоминает запись в базе данных, можно задействовать сокращенный синтаксис конструктора объекта JavaScript:

```
var dataObject = {свойство0:"значение0", свойство1:"значение1"
   свойствоN:"значением"};
```

Формированию массива объектов помогает его свойство length. Благодаря вычисляемым индексам вам не придется вручную перенумеровывать элементы массива после изменения их последовательности:

```
var dataArray = new Array 0;
dataArray[dataArray.length] = {свойство0_0:"значение0_0",
   свойство0_1:"значение0_1",   . свойство0_N:"значение0_N"};
dataArray[dataArray.length] = {свойство1_0:"значение1_0",
   свойство1_1:"значение1_1"      свойство1_N:"значение1_N"}
```

Значения, хранящиеся в массивах или в свойствах объектов, могут быть строками в кавычках (как в показанных примерах), числами, логическими (булевскими) значениями, а также другими массивами и объектами. Количество уровней вложения практически не ограничено, поэтому можно создавать сложные объекты, свойства которых, в свою очередь, будут сложными объектами или массивами.

Обсуждение

Объекты, описанные в сценарии, помещенном в ограниченной тегом <head> области, будут готовы к использованию к тому моменту, когда начнется процесс отображения документа. Таким образом, у вас есть выбор: можно применить обработчик события onload, запускающий функцию формирования содержимого страницы, или же динамически формировать код в процессе загрузки.

Не обязательно напрямую встраивать данные в HTML-страницу. Вместо этого можно задействовать отдельный серверный процесс (запускаемый отдельным URL), формирующий библиотеку сценариев. URL этого серверного процесса нужно указать в атрибуте src тега <script> на странице. Используя такую методику, необходимо убедиться, что возвращаемое серверным процессом содержимое действительно содержит только текст (как обычный .js файл) и задан тип содержимого text/javascript. Из-за потенциальных проблем с синхронизацией вторич-

ного запроса к серверу доставленные данные нельзя применять при динамической генерации HTML.

Одним из самых удобных форматов данных JavaScript является объект, в котором можно применить строковые индексы. Они по своей сути не являются массивами, так как не имеют свойства length, а для перебора элементов необходимо применять цикл типа for-in. Малоизвестен тот факт, что JavaScript позволяет строить такую имитацию хэш-таблицы поверх существующего объекта массива, и при этом два набора данных не будут конфликтовать (см. рецепт 3.9). Рассмотрим следующий пример, в котором результаты Чемпионата мира хранятся в виде массива объектов:

```
var matchData - new Array();
matchData[0] - {loser:"Argentina". lossscore:"2". location:"Urugvay".
winner:"Urugvay". winscore:"4". year:"1930"};
matchData[1] - {loser:"Czechoslovakia". lossscore:"1". location:"Italy".
winner:"Italy". winscore:"2". year:"1934'};
```

А теперь представим себе страницу, содержащую элемент select, позволяющий посетителю выбирать год, за который он хочет увидеть результаты. Медленный способ — с помощью цикла for перебирать все записи массива matchArray, выбирая те, в которых значение свойства year совпадает с выбранным. Если массив имеет большой размер, этот процесс может занять несколько секунд. Но можно сначала однократно создать хэш-таблицу, в которой года будут строковыми индексами. Это можно сделать так (сразу после заполнения массива):

```
for (var i - 0: i < matchData.length: i++) {
   matchData["_" + matchData[i] year] = matchData[i]:
}
```

Обратите внимание на интересную уловку, которая нужна при использовании предложенных данных: если взять год как индекс, его значение будет автоматически преобразовано в число, поэтому спереди к году приписывается нечисловое значение (знак подчеркивания), чтобы сделать его строковым. Без этого к массиву просто будут добавляться новые записи, не попадая в хэш-таблицу. Если же индексами являются нечисловые значения, эта уловка не нужна.

Создав хэш-таблицу, можно обращаться к массиву, используя строковый индекс, как в этом (упрощенном) примере:

```
<select onchange="alert('Победитель: ' + matchData[this.value].winner)">
<option value-"_1930">1930</option>
<option value="_1934">1934</option>

</select>
```

Смотрите также

Как создавать массивы, показано в рецепте 3.1. Создание объектов описывается в рецепте 3.8. Как создавать имитированные хэш-таблицы для массивов объектов, рассказано в рецепте 3.9. Как сортировать массивы объектов, описано в рецепте 3.11.

14.6. Преобразования XML в HTML-таблицы

NN6 IE5(Win)

Задача

Данные XML необходимо отобразить в виде обычной HTML-таблицы.

Решение

Для таблицы с известными заголовками и известной структурой данных XML необходимо создать на странице исходный элемент table, подготовив его к добавлению новых строк с данными:

```
<table id="cupFinals">
<thead>
<tr><th>Год</th>
    <th>Страна проведения</th>
    <th>Победитель</th>
    <th>Проигравший</th>
    <th>Счет (победа - проигрыш)</th>
</tr>
</thead>
<tbody id="matchData"></tbody>
</table>
```

Для загрузки XML можно применить предложенный в рецепте 14.4 сценарий. Функция инициализации страницы (вызываемая из обработчика события onload) вызывает **verifySupport()**, чтобы убедиться, что браузер поддерживает загрузку XML, а затем использует для формирования таблицы показанную в листинге 14.1 функцию **drawTable()** (вызывая ее через **setTimer()**):

```
// начальная инициализация - вызывается из onload
function init(xFile) {
    // убеждаемся, что браузер загрузил XML-файл
    if (verifySupport(xFile)) {
        // Даем файлу время на загрузку
        setTimeout("drawTable('matchData')", 1000);
    }
}

<body onload="init('worldCupFinals.xml')">
```

Обсуждение

В листинге 14.1 приведена функция **drawTable()**, которая преобразует структуру узлов XML в строки таблицы.

Листинг 14.1. Функция **drawTable()**, динамически заполняющая таблицу

```
// Формирование таблицы на основе данных в xDoc
function drawTable(tbody) {
    var tr, td, i, j, oneRecord;
    tbody = document.getElementById(tbody);
```

продолжение

Листинг 14.1 (продолжение)

```
    // дерево узлов
    var data - xDoc.getElementsByTagName( worldcup")[0];
    // для атрибутов класса td
    var classes - ["ctr","","",.....ctr"]·
    for (i - 0: i < data.childNodes.length: 1++) {
        // use only 1st level element nodes to skip 1st level text nodes 1n NN
        // используем только узлы злеиентов первого уровня  пропуская
        // текстовые узлы первого уровня в NN
        if (data.childNodes[1] nodeType -- 1) {
           // одна из записей матча
            oneRecord = data.childNodes[i],
            tr - tbody.insertRow(tbody.rows.length);
            td - tr.insertCell(tr.cells.length);
            td.setAttribute("class",classes[tr.cells.length-1]);
            td.innerHTML - oneRecord.getElementsByTagName("year")[0].
              firstChild.nodeValue:
            td - tr.insertCell(tr.cells.length):
            td.setAttribute("class",classes[tr.cells.length-1]);
            td.innerHTML - oneRecord.getElementsByTagName("location")[0]
              firstChild.nodeValue
            td - tr.insertCelKtr. cells. length);
            td.setAttribute("class",classes[tr.cels. length-1]):
            td.innerHTML - oneRecord.getElementsByTagName("winner")[0].
              firstChild.nodeValue;
            td - tr.insertCell(tr.cells.length):
            td.setAttribute("class",classes[tr.cells.length-1]);
            td.innerHTML = oneRecord.getElementsByTagName("loser")[0].
              firstChild.nodeValue:
            td = tr.insertCell(tr.cells.length).
            td.setAttribute("class",classes[tr.cells.length-1]);
            td.innerHTML - oneRecord.getElementsByTagName("winscore")[0]
              firstChild.nodeValue + " - " +
              oneRecord.getElementsByTagName("losscore")[0].
              firstChild.nodeValue:
        }
    }
}
```

После того как drawTable() закончит работу, на странице появляется таблица, показанная на рис. 14.1.

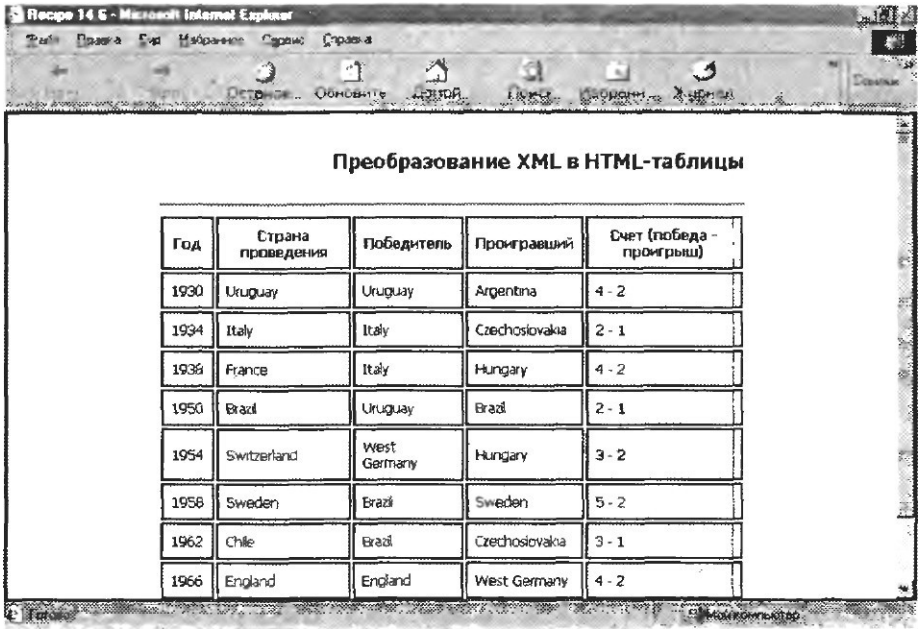
Регулярная структура дерева узлов XML представляет точный аналог форматирования строк и ячеек HTML-таблицы. Начав с общего контейнера (в нашем примере это worldcup), относительно несложно перебрать все элементы следующего уровня, представляющие собой отдельные записи.

Код в листинге 14.1 использует для вставки строк и ячеек в таблицу методы W3C DOM. Оформление задает таблица стилей, расположенная в головной секции документа, она задает правила для некоторых компонентов таблицы, а также для класса ctr:

```
<style type="text/css">
table {table-collapse:collapse: border-spacing·0}
td {border:2px groove black: padding:7px: background-color:#ccffcc]
```

```
th {border:2px groove black: padding:7px; background-color:#ffffcc}
 ctr {text-align:center}
</style>
```

Немного выше функции drawTable() расположен массив, названный classes. Каждая запись этого массива соответствует одному столбцу таблицы. Атрибуту class отдельных столбцов следует присвоить значение ctr. Назначение имени класса производится в цикле, так как метод **setAttribute()** необходимо вызывать для каждой ячейки в отдельности. Если впоследствии вам придется изменить поведение отдельного столбца, нужно будет просто задать правило для этого класса и поместить имя этого класса в конструктор массива classes.



Год	Страна проведения	Победитель	Проигравший	Счет (победа - проигрыш)
1930	Uruguay	Uruguay	Argentina	4 - 2
1934	Italy	Italy	Czechoslovakia	2 - 1
1938	France	Italy	Hungary	4 - 2
1950	Brazil	Uruguay	Brazil	2 - 1
1954	Switzerland	West Germany	Hungary	3 - 2
1958	Sweden	Brazil	Sweden	5 - 2
1962	Chile	Brazil	Czechoslovakia	3 - 1
1966	England	England	West Germany	4 - 2

Рис. 14.1. Пример таблицы, построенной на основе данных XML

Для заполнения таблицы не обязательно применять свойство **innerHTML**. Добавив несколько новых строк, можно получить код, полностью соответствующий W3C DOM. Замена должна выглядеть так (предполагается, что задана еще одна локальная переменная, txt):

```
txt - document.createTextNode(oneRecord getElementsByName("year")[0].
   firstChild.nodeValue);
td.appendChild(txt):
```

Если бы последний столбец таблицы не совмещал данные из нескольких свойств, можно было бы написать универсальную функцию преобразования XML в HTML, которая бы формировала даже шапку таблицы. Текст заголовков можно было бы формировать из имени тега, сделав первую букву заглавной (из соображений эстетики). Конечно, так можно делать только в том случае, если теги имеют осмысленные имена.

Смотрите также

В рецепте 14.4 показано, как внедрять внешние XML-данные в страницу. Преобразование XML в объекты JavaScript демонстрируется в рецепте 14.8. Некоторые подробности о том, как обходить дерево узлов документа, приведены в рецепте 14.17.

4.7. Преобразование данных JavaScript в HTML-таблицы

NN6 IE5(Win)

Задача

Необходимо отобразить внедренные в страницу данные JavaScript в виде HTML-таблицы.

Решение

Построить таблицу в том случае, когда данные представлены в виде массива объектов, сравнительно несложно. Ниже приведен пример данных с регулярной структурой, оформленных в виде массива JavaScript (например, это может быть результат запроса к базе данных):

```
// Табличные данные в виде массива объектов
var jsData = new Array();
jsData[0] = {location:"Uruguay". year:1930. winner:"Uruguay". winScore:4.
             loser:"Argentina". losScore:2}:
jsData[1] = {location:"Italy". year:1934. winner:"Italy", winScore:2.
             loser:"Czechoslovakia". losScore:1}:
jsData[2] - {location."France". year:193B. winner:"Italy", winScore:4,
             loser:"Hungary". losScore:2}:
jsData[3] - {location:"Brazil". year:1950. winner:"Uruguay". winScore:2.
             loser:"Brazil". losScore:1}:
jsData[4] - {location:"Switzerland". year:1954, winner:"West Germany".
           winScore:3. loser:"Hungary". losScore:2}:
```

Скелет таблицы задается в HTML-коде, поэтому заголовки столбцов уже на своих местах, а тело таблицы готово к заполнению динамически генерируемым содержимым:

```
<table id="cupFinals">
<thead>
<tr><th>Год</th>
    <th>Место проведения</th>
    <th>Победитель</th>
    <th>Проигравший</th>
    <th>Счет (победа - поражение)</th>
</tr>
```

```
</thead>
<tbody id="matchData"></tbody>
</table>
```

Функция, заполняющая таблицу данными, использует методы W3C DOM для работы с таблицами:

```
// Построение таблицы на основе данных из массива объектов 'jsData'
function drawTable(tbody) {
    var tr, td;
    tbody = document.getElementById(tbody);
    // цикл по всем записям источника данных
    for (var i = 0; i < jsData.length; i++) {
        tr = tbody.insertRow(tbody.rows.length);
        td = tr.insertCell(tr.cells.length);
        td.setAttribute("align", "center");
        td.innerHTML = jsData[i].year;
        td = tr.insertCell(tr.cells.length);
        td.innerHTML = jsData[i].location;
        td = tr.insertCell(tr.cells.length);
        td.innerHTML = jsData[i].winner;
        td = tr.insertCell(tr.cells.length);
        td.innerHTML = jsData[i].loser;
        td = tr.insertCell(tr.cells.length);
        td.setAttribute("align", "center");
        td.innerHTML = jsData[i].winScore + " - " + jsData[i].losScore;
    }
}
```

Вызывать функцию drawTable() можно из обработчика события onload или из сценария, включенного в тело страницы где угодно после таблицы:

```
drawTable("matchData");
```

Обсуждение

W3C DOM включает в себя ключевые методы для работы с таблицами «на лету». Первый из них — метод insertRow(), имеющийся у таблицы. Второй метод, insertCell(), принадлежит объекту, представляющему строку таблицы. Хотя обычно достаточно только добавить в таблицу строку. Кроме того, добавленные в таблицу ячейки следует заполнить содержимым.

Метод insertRow() возвращает ссылку на только что созданный объект - строку таблицы. Для того чтобы добавить ячейки, необходимо применить эту ссылку, вызывая для создания каждой ячейки метод insertCell(). Оба упомянутых метода требуют один параметр, целое число, означающее положение новой строки в таблице или ячейки в строке. В этом примере применяется простая методика определения положения новой строки, использующая свойство length.

Не обязательно задействовать для заполнения ячеек таблицы свойство innerHTML Добавив в код несколько строк, можно сделать его полностью совместимым со стандартом W3C DOM. Присваивание значения свойству innerHTML тогда

можно заменить таким кодом (должна быть задана дополнительная локальная переменная):

```
txt - documnt.createTetNode(oneRecord.getElementsByTagName("year")[0].
   firstChild.nodeValue);
td.append(txt):
```

Конечно, не обязательно применять данные JavaScript для формирования таблиц. Объекты данных можно использовать как эквивалент таблиц подстановки, взаимодействующих с полями ввода на странице. В рецепте 14.5 приведен пример элемента select, служащего для поиска данных. Передав клиенту возможности решать некоторые несложные задачи поиска и подстановки, можно разгрузить сервер для выполнения более важных обязанностей.

Учтите, что IE для Macintosh не поддерживает эти методы модификации таблиц. Другие методики изменения структуры документа, вовлекающие таблицы или их компоненты, практически всегда приводят к сбою этого браузера.

Смотрите также

Как встроить данные JavaScript в страницу, рассказано в рецепте 14.5. Сортировка таблиц обсуждается в рецепте 14.16.

14.8. Преобразование XML в объекты JavaScript

NN6 IE5(Win)

Задача

Необходимо преобразовать данные XML (загруженные из внешнего файла или внедренные в страницу в виде островка XML в IE/Windows) в объекты JavaScript для дальнейшей обработки.

Решение

Приведенная ниже функция XML2JS() предполагает регулярную, состоящую из отдельных записей структуру данных XML (или той части XML, которая важна для страницы). Функции требуется два аргумента: ссылка на виртуальный документ XML (см. рецепт 14.4) и имя тега элемента XML, родительского для повторяющихся записей:

```
// преобразование XML в массив объектов JavaScript
function XML2JS(xmlDoc, containerTag) {
    var output - new Array();
    var rawData - xmlDoc.getElementsByTagName(containerTag)[0];
    var i. J, oneRecord. oneDbject:
    for (i - 0: i < rawData.childNodes.length; i++) {
        if (rawData.childNodes[i].nodeType — 1) {
            oneRecord = rawData.childNodes[i];
```

```
            oneObject - output[output.length] = new Object().
            for (j = 0; j < oneRecord.childNodes.length: j++) {
                if (oneRecord.childNodes[j].nodeType -- 1) {
                    oneObject[oneRecord.childNodes[j].tagName] =
                        oneRecord.childNodes[j].firstChild nodeValue.
                }
            }
        }
    }
    return output
}
```

Функция возвращает массив объектов, в которых имена свойств совпадают с именами XML-тегов, а значения взяты из текстовых узлов этих тегов.

Обсуждение

В качестве примера применения приведенной в решении функции рассмотрим XML-файл с результатами финала Чемпионата мира, показанный в обсуждении рецепта 14.4. Родительским элементом для отдельных записей в нем является тег worldcup, имя этого тега нужно передать в качестве второго параметра при вызове функции XML2JS(). Результат следует сохранить в переменной:

```
var matchData - XML2JS(xDoc, "worldcup");
```

В результате будет сформирован массив объектов. Первый объект этого массива будет иметь такие значения, как если бы для его создания был использован конструктор:

```
matchData[0] - {loser:"Argentina", lossscore:"2", location:"Uruguay",
               winner:"Uruguay", winscore:"4", year:"1930"};
```

Преобразованные в формат JavaScript данные предоставляют большую гибкость при отображении в HTML. Существенно упрощают и ускоряют сортировку данных по любому из свойств встроенные в JavaScript возможности сортировки массивов. Если вы отображаете данные в виде таблицы, можно применить рецепт 14.6, демонстрирующий, как можно мгновенно менять таблицу, вместо того чтобы посылать на сервер новый запрос.

Вы можете поинтересоваться, зачем использовать преобразование XML в JavaScript, если можно напрямую преобразовать XML в HTML-код (рецепт 14.6). Преимущество в том, что хранящиеся в массиве объектов данные гораздо проще использовать в сценарии. Более того, можно задействовать мощные процедуры сортировки массивов, применять которые к данным, хранящимся в узлах документа, было бы очень неудобно. Примером может служить создание сортируемых по нескольким столбцам таблиц.

Смотрите также

Как загрузить XML, рассказано в рецепте 14.4. Преобразование XML в HTML-таблицу рассматривается в рецепте 14.6. Сортировка динамически генерируемых таблиц продемонстрирована в рецепте 14.16. Как сортировать массив объектов, описано в рецепте 3.11.

4.9. Создание элементов

NN6 IE5

Задача

Необходимо создать новый элемент и вставить его в текущую страницу.

Решение

В IE 5 и старше, а также в Netscape 6 и старше объект document поддерживает метод W3C DOM для создания новых элементов. Единственный параметр этого метода — строка, содержащая имя тега создаваемого элемента:

```
var elem = document.createElement("p");
```

После создания элемента можно задать значения его свойств и вставить его в документ.

Обсуждение

Передаваемый метод имя тега может быть как в верхнем, так и в нижнем регистре, но, в соответствии со сложившейся в HTML практикой, предпочтительно применять нижний регистр. В любом случае, регистр не влияет на значение свойства tagName созданного элемента, которое всегда в верхнем регистре.

Метод createElement() создает элемент в памяти браузера, но он не становится частью документа. Поэтому необходимо вызвать один из методов модификации дерева узлов документа у элемента, который должен стать родительским:

- appendChild(новыйЭлемент) — этот метод добавляет элемент новыйЭлемент в конец списка потомков; возвращаемое значение представляет собой ссылку на новый узел документа;
- insertBefore(новыйЭлемент, предыдущийУзел) — добавляет элемент новыйЭлемент в список потомков перед элементом предыдущийУзел. Возвращает ссылку на новый узел документа;
- replaceChild(новыйЭлемент, старыйУзел) — замещает узел старыйУзел элементом новыйЭлемент; возвращает ссылку на удаленный узел.

Вот типичный пример последовательности команд, создающих элемент, устанавливающих значения его атрибутов и вставляющих его в документ:

```
var elem = document.createElement("img");
elem.setAttribute("src", "images/logo.jpg");
elem.setAttribute("height", "40");
elem.setAttribute("width", "120");
elem.setAttribute("alt", "Логотип GiantCo");
document.body.appendChild(elem);
```

Здесь для задания значений атрибутов, обычно устанавливающихся в свойствах тега, применяется метод setAttribute(). Для задания свойств можно использовать и соответствующие свойства элемента, но W3C DOM рекомендует применять для

этой цели setAttribute(). Какую методику выбрать, зависит только от стиля программирования, потому что браузеры одинаково воспринимают оба варианта синтаксиса (к тому же, вариант с присваиванием значений свойствам более компактен).

Не обязательно устанавливать атрибуты элемента до того, как он будет вставлен в документ. Опять же, это вопрос стиля, хотя обычно свойства элемента настраиваются перед тем, как помещать его в страницу. В ряде редких случаев, чтобы получить доступ к некоторым свойствам объекта, он должен стать частью структуры документа, так как для назначения значения свойства необходим контекст (см. рецепт 14.12).

Internet Explorer, начиная с 4-й версии, также поддерживает ряд методов для формирования новых элементов и внедрения их в документ. Предлагаемая IE методика ориентирована не на узлы документа, как методика W3C DOM, а на строковое представление HTML. Вместо того чтобы создавать новый объект элемента, в IE следует просто описать элемент в виде строки HTML-кода, а затем вставить этот код в нужное место документа. Используя такую методику, приведенный выше фрагмент кода можно переписать так (код работает только в IE):

```
var elem - "<img src='images/logo.jpg' height='40' width='120' " +
   "alt='Корпорация GiantCo'>";
document.body insertAdjasentHTML("BeforeEnd", elem);
```

Первый параметр метода insertAdjasentHTML() указывает браузеру, что новую строку следует вставлять в самый конец документа, сразу перед закрывающим тегом </body>. При использовании этого метода браузер интерпретирует теги как разметку HTML, а не как обычный текст.

Понятно, что строковый подход во многом проще, особенно если вы уже много лет используете его для программирования в IE. Но стандарт W3C DOM Level 2 не предоставляет (и, похоже, не предоставит в будущем) удобных способов для работы с HTML в строковой форме. Пример того, как один из таких разработанных для IE вариантов синтаксиса попал в Mozilla, смотрите в рецепте 14.10.

14.10. Заполнение нового элемента текстом

NN6 IE5

Задача

Необходимо с помощью сценария сформировать часть тела страницы уже после того, как она загружена.

Решение

W3C DOM предлагает для создания текстовых узлов метод объекта document. Единственный параметр этого метода — строка текста, который нужно поместить в указанный контейнер.

```
var txt - document.createTextNode("Построимся и спасемся.");
```

После того как текстовый узел создан, его можно добавить в любой контейнер, в том числе в тот, который создан, но не внесен в структуру страницы (см. рецепт 14.9).

Обсуждение

Вот типичный пример кода, создающего новый элемент и текст для этого элемента:

```
var elem - document.createElement("p");
var txt - document.createTextNode("Построимся и спасемся.");
elem.appendChild(txt);
document.body.appendChild(elem);
```

Для некоторых элементов требуется несколько уровней вложения, что может усложнить дело, но принцип остается тем же:

```
var myEm  myP, txtl, txt2;
myEm - document.createElement("em");
txtl = document.createTextNode("очень");
myEm .appendChild(txtl);
myP - document createElement("p");
txtl - document createTextNode("Я " ) ;
txt2 = document.createTextNode(' рад тебя видеть.");
myP.appendChild(txtl);
myP.appendChild(myEm);
myP.appendChild(txt2);
document.body.appendChild(myP);
```

В результате выполнения этого кода на странице появится текст, эквивалентный такому HTML:

```
<p>Я <ет>очень</ет> рад тебя видеть.</p>
```

Можно создавать любые комбинации текстовых узлов и элементов, при этом сама организация методов W3C DOM для работы с узлами документа практически всегда гарантирует корректность структуры. Фактически вполне возможно даже добавить два текстовых узла один за другим. Для пользователя такие узлы будут выглядеть как непрерывный текст, а с точки зрения структуры страницы они будут смежными узлами одного типа (3). Если же вы предпочитаете объединять соседние текстовые узлы в один, можно использовать метод normalize() родительского элемента.

С другой стороны, Internet Explorer предлагает два свойства для работы с текстом элемента, получивших широкое распространение в разработках для браузеров этого семейства: innerText и innerHTML. Эти два доступных для чтения и записи свойства позволяют помещать в элемент строку простого или размеченного тегами текста. Если помещать содержащий теги текст в свойство innerHTML, браузер будет интерпретировать эти теги так, как если бы они были доставлены с кодом страницы. Использование же свойства innerText подразумевает простой текст без разметки, то есть теги будут видны пользователям так же, как и остальной текст, с угловыми скобками, именем тега и атрибутами. Основанный на использовании строк подход к управлению содержанием документа

применяется только в модели документа IE (в противоположность модели W3C DOM, полагающейся на древовидную структуру документа).

За прошедшие годы многие сетевые авторы сочли эти свойства настолько удобными, что разработчики Mozilla немного смягчили свои требования строгого соответствия W3C DOM и реализовали поддержку свойства innerHTML. Это удобное свойство позволяет уменьшить размер кода, так как благодаря ему не нужно отдельно создавать текстовые узлы. Например, соответствующий W3C DOM код:

```
var txt - document.createTextNode("Построимся и спасемся.");
document.getElementById("myP") appendChild(txt);
```

может быть заменен таким:

```
document.getElementById("myP").innerHTML - "Построимся и спасемся.";
```

Свойство innerText Mozilla не поддерживает.

Смотрите также

Как создать новый (пустой) элемент любого типа, рассказывается в рецепте 14.9. Рецепт 14.11 рассказывает о применении в качестве временного контейнера для элементов и текстовых узлов объекта DocumentFragment.

14.11. Смешанные элементы и текстовые узлы

NN6 **IE5(Mac)/6(Win)**

Задача

Необходимо сформировать содержимое, состоящее как из элементов, так и из текста внутри них.

Решение

Подходящим контейнером для сборки содержимого может стать объект DocumentFragment, описанный в стандарте W3C DOM:

```
var frag, myEm, txt1, txt2:
frag - document.createDocumentFragment();
myEm = document createElement("em");
txt1 = document.createTextNode("очень");
myEm .appendChild(txt1);
txtl = document.createTextNode("Я ");
txt2 = document.createTextNode(" рад тебя видеть.");
frag.appendChild(txt1).
frag appendChild(myEm);
frag.appendChild(txt2);
```

После выполнения такого кода фрагмент (начинающийся и завершающийся текстовыми узлами) готов для вставки в любой существующий узел документа.

Обсуждение

Объект DocumentFragment можно рассматривать как временный контейнер, черновик, который может содержать любую последовательность узлов любого типа. Этот фрагмент существует только в памяти и не является частью структуры страницы.

В Internet Explorer этот объект поддерживается в версии 5 для Macintosh и в версии 6 для Windows. В более ранних версиях аналогов этому объекту нет, но можно имитировать его, собирая элементы и узлы в любом контейнере (например, в div или span). Затем, когда настанет время поместить собранное содержимое в основной документ, можно последовательно удалить каждый из элементов из временного контейнера и вставить его в документ. Это некрасиво, но вполне возможно.

Если собирать смешанное содержимое не в виде узлов документа, а в виде строк, удобно применить имеющееся у всех элементов (в IE 4 и NN 6) свойство innerHTML. Строковый эквивалент приведенного выше кода выглядит так:

```
var newContent - "Я <em>очень</em> рад тебя видеть.".
```

Чтобы поместить собранное таким образом содержимое в контейнер, необходимо присвоить эту строку свойству innerHTML этого контейнера. В результате новое содержимое заменит в этом контейнере старое.

Объектная модель IE снабжает элементы еще одним методом, с помощью которого можно добавлять в страницу текст, содержащий или не содержащий HTML-разметку. Метод insertAdjacentHTML() (совместимый с IE, начиная с версии 4) позволяет указать, куда будет вставлено содержимое. Этот метод получает два параметра. Первый из них — строка, описывающая относительное положение точки вставки нового содержимого. Вот список значений, которые может принимать этот параметр (регистр важен):

- BeforeBegin — перед открывающим тегом элемента;
- AfterBegin — после открывающего тега, но перед любым другим содержимым контейнера;
- BeforeEnd — перед закрывающим тегом элемента, но после всего его содержимого;
- AfterEnd — сразу после закрывающего тега (перед следующим элементом).

Новое содержимое передается во втором параметре. Например, чтобы присоединить ранее созданную строку HTML-кода к существующему элементу, идентификатор которого myP, в IE нужен следующий обратно-совместимый синтаксис:

```
document.all("myP").insertAdjacentHTML("BeforeEnd". новоеСодержимое);
```

Помимо этого метода IE предлагает широкий диапазон методов управления содержимым, перечисленных в табл. 14.1.

Таблица 14.1. Методы IE для модификации содержимого страницы

Метод	Совместимость	Описание
contains(elemRef)	IE 4	Возвращает true, если элемент содержит в себе elemRef
getAdjacentText(where)	IE 5(Win)	Возвращает текст, содержащийся в where
insertAdjacentElement (where, elemRef)	IE 5(Win)	Вставляет новый элемент в положение, указанное where
insertAdjacentHTML(where, HTMLText)	IE 4	Вставляет в страницу текст (положение задается where), который отображается как HTML
insertAdjacentText(where, text)	IE 4	Вставляет в страницу обычный текст (положение задается where)
removeNode(deep)	IE 4	Удаляет элемент или текстовый узел (а также его потомков, если параметр deep равен true)
replaceAdjacentText(where, text)	IE 5(Win)	Заменяет текст, содержащийся в where, на text
replaceNode(newNodeRef)	IE 5(Win)	Заменяет текущий узел на другой узел
replaceNode(otherNodeRef)	IE 5(Win)	Меняет местами текущий узел и othernodeRef и возвращает ссылку на удаленный элемент

Смотрите также

Как создать новый элемент, рассказано в рецепте 14.9. Рецепт 14.10 описывает создание текстового узла.

14.12. Вставка и заполнение элемента iframe

NN6 **IE5**

Задача

Необходимо создать элемент iframe, вставить его в текущий документ и заполнить содержимым.

Решение

Для решения этой задачи следует не просто создать элемент и присоединить его к странице. Это тот случай, когда один из создаваемых элементов, iframe, должен находиться в контексте документа перед заполнением. Ниже приведен пример кода, рассчитанного на глобальную область видимости. Если вы рассчитываете применить его внутри функции, переменную newIframe необходимо предварительно определить как глобальную.

Итак, сначала присоединим iframe в конец документа, а затем поместим в него динамически генерируемую форму. Начало кода вполне ожидаемо: в памяти создается объект iframe, а затем он помещается в документ:

```
// создаем элемент iframe
var newIframe = document.createElement("iframe")
newIframe.setAttribute("id","newIframe");
// Вставляем созданный узел в документ, чтобы дать ему контекст
// Если узел не нужно видеть, уменьшить его размер или установить
// его свойство display:none
document.body.appendChild(newIframe);
```

Затем нужно получить ссылку на контекст документа, чтобы создавать новые элементы, помещаемые в iframe. Для IE подходящим контекстом является документ внутри iframe, а в других браузерах подходит основной документ:

```
// получаем подходящий контекст для создания элементов
if (navigator.appName == "Microsoft Internet Explorer") {
    var doc - newIframe.contentWindow.document;
} else {
    doc = document;
}
```

Теперь, используя этот контекст, создадим форму и элемент управления внутри нее. К сожалению, IE зачастую «бежит впереди себя», в результате чего операторы могут выполняться до того, как возникнет создаваемый объект. Поэтому нужно притормозить выполнение с помощью метода setTimeout() (установив нулевую задержку), чтобы действия выполнялись в стабильном окружении:

```
// создаем форму в подходящем контексте
var newForm - doc.createElement("form")
newForm.setAttribute("id","sendform");
// в том же контексте создаем поле ввода
var newField = doc.createElement("input");
newField.setAttribute("id","alldata");
newField.setAttribute("type","text");
// вставляем поле ввода в форму
newForm.appendChild(newField);
// здесь создаем остальные элементы формы и добавляем их в нее

// помещаем форму в iframe (с задержкой)
setTimeout("finishIframe()" 0);
}
```

Наконец, форма должна быть помещена в документ:

```
// завершаем вставку формы в iframe
function finishIframe() {
    newIframe.contentWindow.document.body.appendChild(newform);
```

Обсуждение

Если записать в виде обычной страницы HTML-код для элемента iframe и его содержимого, элементы формы просто будут помещены между открывающим

и закрывающим тегами iframe. Это может навести на мысль, что создать новую форму и поместить ее в контейнер можно точно так же, как в рецептах 14.10 и 14.9. Но различие в том, что iframe — один из вариантов объекта window, как фрейм или набор фреймов. Следовательно, чтобы сценарий мог работать с содержимым этого объекта, он должен обращаться к находящемуся у него внутри документу. Если сценарий выполняется в контексте главного документа и все, что имеется, — ссылка на iframe, путь к содержащемуся в нем документу открывает свойство contentDocument, хранящее ссылку на соответствующее этому элементу окно. Причем это окно не существует в объектной модели до тех пор, пока iframe не помещен в документ.

Такой окольный способ доступа к содержимому применим только к элементам, содержащим объект window или document: frame и iframe. Элемент object также содержит ссылки на другие объекты, но окна в них не вовлечены. Вместо этого у элемента object имеется свойство contentDocument (есть в W3C DOM, но не поддерживается IE 6). К остальным HTML-элементам описанные сложности не относятся.

Смотрите также

Как объединить несколько отдельных HTML-документов, используя iframe, рассказано в рецепте 14.3. Создание новых элементов документа рассматривается в рецепте 14.9. Формирование текстового содержимого объектов описывается в рецепте 14.10.

14.13. Как получить ссылку на HTML-элемент

NN6 IE5

Задача

Зная идентификатор или имя тегу существующего элемента, необходимо получить на него ссылку.

Решение

Б W3C DOM имеется метод, позволяющий получить ссылку на любой объект в документе:

```
var elem = getElementById("идентификаторЭлемента");
```

Если элементу не назначен идентификатор, можно обратиться к нему по имени тега. Следующий пример формирует массив объектов, имеющих одинаковое имя тега:

```
var elems = getElementsByTagName("имяТега");
```

Получив массив ссылок и зная положение искомого элемента среди остальных, для получения нужной ссылки следует использовать стандартный синтаксис для работы с массивами:

```
var elem - getElementsByTagName("имяТега")[2];
```

Обсуждение

Метод getElementById() принадлежит объекту document, поэтому областью его действия всегда является документ целиком, включая головную секцию. В противоположность ему, метод getElementsByTagName() имеется у любого контейнера, благодаря этому можно уточнить область рассмотрения. Кроме того, в IE 6, NN 6 и более новых версиях можно получить массив всех элементов с помощью группового символа звездочки:

```
var allElems - document.getElementByTagName("*");
```

Если сценарий должен работать с группами однотипных элементов, идентификаторы следует назначать так, чтобы упростить работу циклов for. Например, если сценарий используется для динамического формирования таблицы, ее ячейкам можно назначать последовательные идентификаторы, такие как subTotalO, subTotall и т. д. Если позже понадобится перебирать ячейки, можно избежать использования чудовищно неэффективной функции eval(), получая идентификатор для getElementById() с помощью операции склеивания строк (конкатенации):

```
for (var i - 0: i < array.length; i++) {

    subTotCell - document.getElementById("subTotal" + i);

}
```

Если вам необходимо получить ссылку на элемент в одном из ранних браузеров, можно воспользоваться парой функций DHTML API из рецепта 13.3, работающих в браузерах разных версий:

```
// Поиск слоя (layer) NN4 по его строковому имени
function seekLayer(doc, name) {
    var theObj;
    for (var i - 0: i < doc.layers.length; i++) {
        if (doc.layers[i].name -- name) {
            theObj - doc.layers[i]:
            break;
        }
       // переход к вложенным слоям
        if (doc.layers[i].document.layers.length > 0) {
            theObj - seekLayer(document.layers[i].document, name);
        }
    }
    return theObj:
}
// Преобразование строки с именем обьекта или ссылки на объект
// в ссылку на обьект.
function getRawObject(obj) {
    var theObj:
```

```
    if (typeof obj — "string") {
        if (isW3C) {
            theObj - document.getElementById(obj);
        } else if (isIE4) {
            theObj = document.all(obj);
        } else if (isNN4) {
            theDbj - seekLayer(document. obj);
        }
    } else {
       // если получена ссылка - передаем ее без изменений
        theObj - obj:
    }
    return theObj:
}
```

Чтобы использовать эти функции, необходимо вызвать **getRawObject(),** передав ей строку с идентификатором нужного элемента. В IE 4 функция применяет коллекцию **document.all,** содержащую все элементы документа. В NN 4 таким образом можно обращаться только к позиционируемым слоям (к тому же в NN 4 не все элементы представлены в объектной модели). В тех ситуациях, когда один слой вложен в другой, функция **getRawObject()** задействует функцию seekLayer(), перебирающую все слои в поисках одного из них, имя которого совпадает со значением искомого идентификатора.

Смотрите также

В рецепте 13.3 смотрите полное описание того, как библиотека DHTML API применяет для позиционирования настраиваемые в зависимости от платформы ссылки на объекты.

14.14. Замена части содержимого

NN6 IE5

Задача

Необходимо заменить содержимое тела документа на динамически сгенерированное.

Решение

Существует несколько подходов, применять которые следует в зависимости от того, нужно ли заменить содержимое целиком или следует менять текст в группе HTML-элементов. Если необходимо заменить текст внутри контейнера, создайте новый текстовый узел и замените им текущий дочерний узел контейнера:

```
var txt = document.createTextNode("Построимся и спасемся!");
var elem - document.getElementById("someElement");
var oldTxt - elem.replaceChild(txt. elem.firstChild);
```

Если контейнер содержит не только текст, но и перекрывающиеся элементы (например, абзац (p), часть текста в котором выделена тегами em), то перед тем, как помещать в него новый текст, удалите все дочерние узлы:

```
var txt = document.createTextNode("Построимся и спасемся!");
var elem = document.getElementById("someElement");
while (elem.childNodes.length > 0) {
    elem.removeChild(elem.firstChild);
}
elem.appendChild(txt);
```

Чтобы заменить один из дочерних объектов сгенерированным текстом, применяется метод replaceChild():

```
var newElem = document.createElement("span");
newElem.setAttribute("id", "newSpan");
var elem = document.getElementById("someElement");
elem.replaceChild(newElem, elem.firstChild);
```

В более сложных случаях, особенно когда новое содержимое начинается или заканчивается текстовым узлом, в качестве временного контейнера для его сборки используйте объект DocumentFragment.

Обсуждение

Если вам необходимо заменить только часть текста в узле, у вас есть несколько вариантов на выбор. Самый запутанный из них вовлекает объекты текстовых диапазонов, показанные в рецепте 15.3. Однако в менее сложных случаях можно применить несложные процедуры анализа исходного и нового текстов. Простейшая последовательность действий включает три шага. Сначала из элемента с помощью свойства nodeValue извлекается текст. Затем, чтобы заменить старую подстроку на новую, вызывается имеющийся у строки метод replace(). После чего новый текст помещается в текстовый узел на место старого. Ниже приведен короткий пример, заменяющий строку "coming" на "going":

```
var elem = document.getElementById("myP");
var srchText = /coming/g;
var replacement = "going";
var elemText = elem.firstChild.nodeValue.replace(srchText, replacement);
elem.firstChild.nodeValue = elemText;
```

Если используемый дизайн позволяет заранее знать, какая именно часть текста должна быть замещена, проще всего обернуть текстовый фрагмент в span, с помощью которого затем можно будет локализовать нужную часть текста. Аналогичным образом можно создать страницу, некоторые части которой изначально пусты и заполняются текстом после загрузки или в результате взаимодействия с пользователем. Для этого в нужные места страницы следует вставить пустые элементы. Примеры вы можете увидеть в рецептах 14.6 и 14.7, где в таблицу добавлен пустой элемент tbody, в который после загрузки будут помещены сгенерированные сценарием строки и ячейки таблицы.

Смотрите также

Создание элементов рассматривается в рецепте 14.9. Формирование текстового содержимого описывается в рецепте 14.10. Дальнейшие сведения об объекте DocumentFragment можно увидеть в рецепте 14.11. Поиск и замена в строках с помощью регулярных выражений описаны в рецепте 1.7. В рецепте 15.3 рассказывается, как использовать текстовые диапазоны в операциях поиска и замены.

14.15. Удаление части страницы

NN6 IE5

Задача

Нужно исключить из текущей страницы элемент или часть текста.

Решение

Если имеется ссылка на элемент, который нужно удалить, можно воспользоваться методом W3C DOM removeChild(). Этот метод применим только к дочерним узлам, поэтому, чтобы его вызвать, необходимо подняться на один уровень вверх и получить ссылку на родительский элемент:

```
var elem - document.getElementById("spanToGo");
elem.parentNode.removeChild(elem);
```

Чтобы убрать из текстового узла текст, следует поместить в этот узел пустую строку:

```
var container = document.getElementById("someSpan");
// проверка того, что узел действительно текстовый
if (container.firstChild.nodeType == 3) {
    container.firstChild.nodeValue = ""
}
```

Чтобы полностью удалить текстовый узел, можно, как это было показано выше, воспользоваться методом removeChild().

Обсуждение

Если вы удаляете последовательные группы элементов, например строки таблицы, для их перебора обычно можно применить соответствующую коллекцию. Например, у объектов table или tbody имеется свойство rows, возвращающее коллекцию всех вложенных элементов tr. Если вы собираетесь удалить все строки таблицы, можно сделать это с помощью цикла while, удаляя первого потомка до тех пор, пока не останется ни одного:

```
var tbody = document.getElementById("myTableBody");
while (tbody.rows.length > 0) {
    tbody.removeChild(tbody.firstChi ld);
}
```

Но если удалять нужно выборочно, необходимо обязательно учитывать изменение нумерации элементов коллекции по мере того, как массив уменьшается. Чтобы обойти возможные проблемы, используйте цикл for, начиная перебор с конца и уменьшая значение индекса при каждом шаге. Благодаря этому цикл не выйдет за пределы коллекции.

Чтобы продемонстрировать работу выборочного удаления, рассмотрим следующую таблицу, в каждой строке которой имеется флажок и некоторый текст. Внизу таблицы имеется кнопка, после щелчка на которой все строки, в которых флажок отмечен, удаляются. HTML-код таблицы выглядит так:

```
<form>
<table>
<tbody id="myTBody">
<tr>
    <td><input type="checkbox"></td><td>Пункт 1</td>
</tr>
<tr>
    <td><input type="checkbox"></td><td>Пункт 2</td>
</tr>
<tr>
    <td><input type="checkbox"></td><td>Пункт 3</td>
</tr>
<tr>
    <td><input type="checkbox"></td><td>Пункт 4</td>
</tr>
<tr>
    <tdxi nput type="checkbox"></td><td>Пункт 5</td>
</tr>
<tr>
    <td colspan="2">
    <input type="button" value="Удалить выделенное" onclick="remove()"></td>
    </td>
</tr>
</tbody>
</table>
</form>
```

Тогда функция remove() должна выглядеть так:

```
function remove() {
    var elem - document.getElementById("myTBody");
    for (var i - elem.rows.length-1; i >= 0 : i-) {
        if (elem.rows[i].cells[0].firstChild.checked) {
            elem.removeChild(elem.rows[i]);
        }
    }
}
```

Кроме того, чтобы удалять строки таблицы, в этой функции можно было бы применить метод deleteRow()

Смотрите также

Как получить ссылку на нужный элемент страницы, рассказывается в рецепте 14.13.

14.16. Сортировка динамических таблиц

NN6 **IE5(Win)**

Задача

Необходимо создать таблицу, которую пользователь мог бы сортировать по разным колонкам.

Решение

Проще всего сортировку таблиц реализовать в том случае, когда данные доставляются в виде XML или JavaScript, как в рецептах 14.6 и 14.7. При этом данные для таблицы можно оставить прежними. Изменения следует внести в код фиксированных заголовков таблицы, также нужно написать функции, вызывающиеся при щелчке по ним.

Структуру HTML-кода страницы разрабатывают так, чтобы дать пользователю возможность управлять сортировкой. Самый распространенный способ — оформить текст заголовков столбцов в виде гиперссылок. Затем нужно определить функции сортировки для каждого возможного порядка сортировки. Пример таких функций вы можете увидеть в обсуждении. Наконец, с помощью сценария следует генерировать таблицу на основе выбранного порядка сортировки (см. обсуждение). Каждый раз, когда пользователь выбирает новый порядок сортировки, тело таблицы обновляется и заполняется данными в желаемом порядке.

Обсуждение

Чтобы показать, на что должна быть похожа основа сортируемой таблицы, приведен HTML-код из рецепта 14.7, в котором изменены только заголовки столбцов (элемент th), в которые добавлены гиперссылки.

```
<table id="cupFinals">
<thead>
<tr>
    <th><a href="#" title="Сортировать по году"
           onclick="return sortTable(this)">Год</a></th>
    <th><a href="#" title="Сортировать по стране"
           onclick="return sortTable(this)">Страна проведения</a></th>
    <th><a href="#" title="Сортировать по победителю"
           onclick="return sortTable(this)">Победитель</a></th>
    <th><a href="#" title="Сортировать по проигравшему"
           onclick="return sortTable(this)">Проигравший</a></th>
    <th>Счет <a href="#" title="Сортировать по счету"
                onclick="return sortTable(this)">победа</a>  <a href="#"
                title="Sort by Losing Score"
                onclick="return sortTable(this)">поражение</a></th>
</tr>
</thead>
<tbody id="matchData"></tbody>
</table>
```

Все гиперссылки вызывают одну и ту же функцию **sortTable()**, работающую как коммутатор для вызова отдельных процедур сортировки. В зависимости от элемента th, в котором находится гиперссылка, выбирается один из методов сортировки (с помощью оператора switch):

```
// Диспетчер функций сортировки (вызывается при щелчке на заголовке столбца)
function sortTable(link) {
    switch (link.firstChild.nodeValue) {
        case "Год" :
            jsData.sort(sortByYear):
            break:
        case "Страна проведения" :
            jsData.sort(sortByHost);
            break:
        case "Победитель" :
            jsData.sort(sortByWinner);
            break;
        case "Проигравший" :
            jsData.sort(sortByLoser):
            break:
        case "победа" :
            jsData.sort(sortByWinScore);
            break:
        case "поражение"
            jsData.sort(sortByLosScore);
            break:
    }
    drawTable("matchData")
    return false
}
```

Каждая из отдельных процедур сортировки сортирует массив jsData, основываясь на значении одного из свойств его элемента (элементами этого массива являются объекты):

```
// функции сортировки (вызываются из sortTable())
function sortByYear(a, b) {
    return a.year - b.year:
}
function sortByHost(a. b) {
    a = a.location.toLowerCase();
    b = b.location.toLowerCase();
    return ((a < b) ? -1 : ((a > b) ? 1   0));
}
function sortByWinScore(a. b) {
    return b.winScore - a.winScore:
}
function sortByl_osScore(a. b) {
    return b.losScore - a.losScore:
}
function sortByWinner(a. b) {
    a - a.winner.toLowerCase();
    b - b.winner.toLowerCaseO:
    return ((a < b) ? -1 : ((a > b) ? 1 : 0)):
}
```

```
function sortByLoser(a. b) {
    a = a.loser.toLowerCase();
    b = b.loser.toLowerCase();
    return ((a < b) ? -1    ((a > b) ? 1    0)).
}
```

Вернемся к функции sortTable(). После того как массив jsData отсортирован по одному из признаков, выполняется функция drawTable(). Небольшим отличием от рецепта 14.7 является вызов другой функции, удаляющей все строки из тела таблицы. Вот как выглядит модифицированный метод drawTable():

```
// формирование таблицы на основе масива объектов  jsData'
function drawTable(tbody) {
    var tr. td:
    tbody = document.getElementById(tbody);
    // удаляем существующие строки (если есть)
    clearTable(tbody):
    // цикл по всему источнику данных
    for (var i = 0: i < jsData.length; i++) {
        tr = tbody.insertRow(tbody.rows. length):
        td = tr insertCell(tr. cells. length):
        td.setAttribute("align". "center"):
        td.innerHTML = jsData[i].year:
        td = tr.insertCell(tr.cells.length);
        td.innerHTML - jsData[i].location:
        td = tr.insertCell(tr.cells. length):
        td.innerHTML = jsData[i].winner:
        td = tr.insertCell(tr.cells.length);
        td.innerHTML = jsData[i].loser:
        td = tr.insertCell(tr.cells. length):
        td.setAttribute("align". "center");
        td.innerHTML = jsData[i].winScore + "    " + jsData[i].losScore:
```

Функция clearTable() содержит простой цикл, удаляющий все строки из тела таблицы:

```
// удаление существующих строк
function clearTable(tbody) {
    while (tbody.rows.length > 0) {
        tbody.deleteRow(0);
    }
}
```

Большинство функций сортировки в этом рецепте основываются на значении одного из свойств, хранящихся в массиве объектов. Так как каждый элемент массива jsData — объект (смотрите определение этого объекта в рецепте 14.7), задача функции сортировки — последовательно сравнивать значения отдельных свойств, таких как названия команд или величина счета. Чтобы строки сравнивались без учета регистра символов, перед сравнением они приводятся к одному регистру (в данном случае — к нижнему), так что различие ASCII-кодов строчных и прописных символов не сказывается на результатах сортировки.

Смотрите также

Как внедрять в страницу данные в виде объектов или массивов JavaScript, рассказывается в рецепте 14.7. Простая сортировка массива описывается в рецепте 3.5. Сортировка массива объектов рассматривается в рецепте 3 10.

14.17. Обход узлов документа

NN6 IE5

Задача

Необходимо перебрать все узлы документа, чтобы найти те из них, которые удовлетворяют нужному критерию.

Решение

Показанная ниже функция getLikeElements() возвращает коллекцию элементов, у которых одинаковое имя тега, имя и значение атрибута:

```
function getLikeElements(tagName, attrName, attrValue) {
    var startSet;
    var endSet = new Array();
    if (tagName) {
        startSet = document.getElementsByTagName(tagName);
    } else {
        startSet = (document.all) ? document.all : document.getElementsByTagName("*");
    }
    if (attrName) {
        for (var i = 0; i < startSet.length; i++) {
            if (startSet[i].getAttribute(attrName)) {
                if (attrValue) {
                    if (startSet[i].getAttribute(attrName) == attrValue) {
                        endSet[endSet.length] = startSet[i];
                    }
                } else {
                    endSet[endSet.length] = startSet[i];
                }
            }
        }
    } else {
        endSet = startSet;
    }
    return endSet;
}
```

Обсуждение

При вызове getLikeElements() можно в определенных комбинациях пропускать один или несколько аргументов. Например, если опустить все параметры, функ-

ция возвратит коллекцию всех узлов документа. Указав только первый аргумент (имя тега), можно получить массив элементов с одинаковым именем тега. Если задать только имя тега и имя атрибута, функция вернет массив объектов, имеющих одинаковое имя тега и заданный атрибут, но без учета значения атрибута. Указывая значение атрибута, следует также указывать и его имя. Чтобы пропустить аргумент, предшествующий используемому аргументу, нужно поместить в него пустую строку или значение **null.** Ниже показан пример корректных вызовов **getLikeElements():**

```
var collection = getLikeElements():
var collection = getLikeElements("td"):
var collection = getLikeElements("". "class"):
var collection = getLikeElements("". "class"  "highlight"):
var collection = getLikeElementsC'td". "align"  "center"):
```

Следует с осторожностью применять эту функцию для поиска элементов **input,** имеющих атрибут value. В Netscape будут возвращены только те элементы **input,** для которых значение этого атрибута задано явно, а в IE возвращаются все, так как браузер автоматически устанавливает этот атрибут у таких элементов input, как переключатели или флажки.

Можно использовать перебирающий узлы документа сценарий для того, чтобы построить схему структуры документа. Имеющиеся в объектной модели средства для получения списка всех узлов полностью скрывают их иерархию. Чтобы сохранить и отследить иерархию, можно применить следующую функцию walkChildNodes(), которая формирует строку, отображающую структуру узлов любого элемента. Ссылка на него должна быть передана в первом параметре. Эта функция рекурсивно вызывает сама себя при переходе к каждому из последующих уровней иерархии и использует второй аргумент, чтобы отслеживать уровень вложения.

```
function walkChildNodes(objRef. n) {
    var obj;
    if (objRef) {
        if (typeof objRef == "string") {
            obj = document.getElementById(objRef);
        } else {
            obj - objRef:
        }
    } else {
        obj = (document.body.parentElement) ?
            document.body.parentElement : document.body.parentNode;
    }
    var output = "":
    var indent = "".
    var i. group. txt:
    if (n) {
        for (i = 0: i < n: i++) {
            indent += "+-":
        }
    } else {
        n = 0:
        output += "Child Nodes of <" + obj.tagName .toLowerCase():
        output += ">\n==========\n":
    }
```

```
    group = obj childNodes;
    for (i = 0; i < group.length; i++) {
        output += indent;
        switch (group[i].nodeType) {
            case 1:
                output += "<" + group[i].tagName.toLowerCase();
                output += (group[i].id) ? " ID=" + group[i].id
                output += (group[i].name) ? " NAME=" + group[i] name
                output += ">\n";
                break;
            case 3:
                txt = group[i].nodeValue.substr(0.15);
                output += "[Text:\"" + txt.replace(/[\r\n]/g."<cr>");
                if (group[i].nodeValue.length > 15) {
                    output += "...";
                }
                Output += "\"]\n";
                break;
            case 8:
                output += "[!COMMENT!]\n";
                break;
            default.
                output += "[Node Type = " + group[i].nodeType + "]\n";
        }
        if (group[i].childNodes.length > 0) {
            output += walkChildNodes(group[i]. n+1);
        }
    }
    return output;
}
```

Чтобы с помощью этой функции определить структуру всех узлов документа, вызов должен выглядеть так:

```
walkChildNodes(document.body);
```

Функция показывает имя тега (с идентификатором) для каждого узла элемента и часть текста из текстовых узлов, чтобы отличать их друг от друга. Ниже показан пример того, что эта функция возвращает для тела документа, составленного из документа из рецепта 14.1 и таблицы из обсуждения рецепта 14.15:

```
Child Nodes of <body>

<h1>
+--[Text:"Добро пожалова..."]
<h2>
+--[Мы любим пользователей]
+--<script>
+--[Text:" Windows"]
+--<noscript>
+—[Text:"!"]
<hr>
<form>
+--<table>
+~ +— <tbody ID=myTBody>
+—+—+—<tr>
+--+--+--+--<td>
```

```
+--+--+--+--+--+--<input>
+ -+--+--+--+--<td>
+--+--+--+--+--+--[Text·"Пункт 1"]
+-+--+--<tr>
+--+--+--+--+--<td>
+--+--+--+--+--+--<input>
+--+--+--+--+--<td>
+--+--+--+--+--+--[Text:"Пункт 2"]
+--+--+--+--<tr>
+--+--+--+--+--<td>
+-+--+--+--+--+--<i nput>
+--+--+--+--+--<td>
```

Функцию walkChildNodes() можно применять как инструмент диагностики, в частности, для динамически формируемого содержимого. Если добавить ее в документ, а также добавить временный элемент **textarea,** можно переписать функцию формирования динамического содержимого так, чтобы она в конце вызывала walkChildNodes() и помещала ответ в **textarea.** Это даст возможность более подробно изучить существующую иерархию и сравнить ее с тем, какой она должна быть.

Одна из последних методик, о которых нужно знать, полагается на объект W3C DOM **TreeWalker,** доступный в Netscape 7 и старше (но не доступный в IE 6-й версии). Этот объект представляет собой живой, иерархический список узлов документа, соответствующих заданному при вызове document.createTreeWalker() критерию. Пункты этого списка подчиняются той же иерархии, что и узлы документа, на которые они ссылаются. Метод createTreeWalker() определяет, откуда начинается список и какие узлы исключаются из него путем фильтрования.

Объект **TreeWalker** поддерживает своего рода указатель внутри списка, а методы этого объекта позволяют переходить к следующему или предыдущему узлам (а также к смежным, дочерним или родительским), перемещая указатель в выбранном направлении. Если сценарий изменит документ уже после того, как **TreeWalker** создан, изменение структуры автоматически отобразится в узлах объекта.

Хотя **TreeWalker** полностью пригоден для использования в HTML-документах, он еще полезнее при работе с XML. Например, W3C DOM не предоставляет быстрого способа обратиться ко всем элементам, имеющим атрибут с определенным именем (что легко можно сделать на сервере с помощью стандарта XPath). Но можно создать TreeWalker, который бы указывал только на узлы, имеющие нужный атрибут. Благодаря этому можно с легкостью последовательно перебрать нужные узлы (то есть без использования более сложных сценариев, перебирающих все узлы в поисках необходимого). Например, следующая функция-фильтр пропускает в объект **TreeWalker** только те узлы, которые содержат атрибут **author:**

```
function authorAttrFilter(node) {
    if (node.hasAttribute("author")) {
        return NodeFilter.FILTER_ACCEPT;
    }
    return NodeFilter.FILTER_SKIP;
}
```

Ссылка на эту функция должна передаваться в качестве одного из параметров метода createTreeWalker(). Также при вызове этого метода можно указать, чтобы перечислялись только элементы:

```
var authorsOnly = document.createTreeWalker(document
   NodeFilter.SHOW_ELEMENT, authorAttrFilter, false);
```

Теперь, чтобы обратиться к отдельным элементам из списка, можно применить методы объекта TreeWalker. При вызове метода объект с помощью функции-фильтра последовательно проверяет всех кандидатов и останавливается, когда узел соответствует критерию. После этого, имея ссылку на элемент, можно обращаться к любым его методам и свойствам, независимо от TreeWalker.

Смотрите также

Как формировать большую строку из фрагментов, рассказывается в рецепте 1.1.

14.18. Считывание содержимого документа

NN6 IE5

Задача

Необходимо скопировать часть дерева узлов текущего HTML-документа, включая любые динамические изменения, сделанные сценариями.

Решение

Потребность в таких действиях чаще всего возникает тогда, когда вы меньше всего подготовились к ним, например при отладке. Поэтому, чтобы сделать копию всего HTML-кода страницы, введите в строку адреса браузера следующий URL (в одну строку):

```
javascript: void window.open("","","").document.write("<textarea cols=80
rows=20>" + document.body.parentNode.innerHTML + "</textarea>")
```

В результате будет открыто окно, содержащее единственный элемент textarea, в котором находится HTML-код между тегами <html> и </html> текущей страницы.

Обсуждение

Если вы динамически создаете элементы, зачастую трудно определить, связана ли ошибка отображения с проблемами браузера или проблема в структуре документа, измененной сценариями. Просмотр кода страницы в браузере особой помощи не оказывает, так как обычно при этом отображается состояние страницы в момент ее появления в браузере и не учитываются динамические изменения, сделанные уже после загрузки.

Идеальным способом диагностики было бы выделить копию HTML-кода, как его видит браузер, поместить ее в отдельный документ и затем отдельно загружать его в браузер. Иногда просто взглянув на HTML-код, можно обнаружить такие ошибки, как неправильно закрытые теги контейнеров (эта ошибка чаще встречается при применении строковых методов модификации содержимого в IE), несбалансированные элементы таблиц или отсутствующие атрибуты. Но если даже проблема менее явная, гораздо проще работать со стабильным HTML без сценариев, экспериментируя с тегами и атрибутами, добиваясь правильной работы во всех браузерах. Исправив ошибки, можно вернуться к сценариям и «залатать дыры» в процедурах динамического формирования содержимого.

Показанный в примере псевдо-URL javascript: можно изменить так, чтобы формировать элемент textarea большего или меньшего размера. Когда вы найдете наиболее подходящую комбинацию размеров, добавьте этот URL в Избранное (создайте закладку). Такая закладка будет работать со всеми страницами, доставленными с любого сервера.

Тот же метод можно использовать даже на страницах с фреймами. Следует просто задать ссылку на нужный фрейм:

```
javascript: void window.open("","","").document.write("<textarea cols=80
rows=20>" + top.frames[1].document.body.parentNode.innerHTML +
"</textarea>")
```

Необходимо учитывать, что в тех случаях, когда страница описана как строго соответствующая XHTML, IE для Windows может не создать всю разметку, особенно содержимое элемента head. Вы можете поэкспериментировать с кодом ссылки, выделяя не весь документ, а отдельные его сегменты.

Смотрите также

В рецепте 14.17 предлагается другой способ исследования структуры документа путем обхода дерева его узлов.

15 Приложения DHTML

15.0. Вступление

Приведенные в этой главе рецепты должны помочь вам решить практические задачи, встречающиеся при разработке DHTML-страниц. Сложность при составлении подобного реестра решений в том, что DHTML достаточно гибок и воображение разработчиков может двигаться в совершенно произвольном направлении. Хотя большинство рецептов можно использовать в готовом виде, они должны служить базовой основой, на которой будет построено конкретное приложение. Если эти рецепты подскажут вам, как улучшить сайт, — тем лучше.

Несколько рецептов базируются на программных объектах, возможности которых не всегда просто применить: объекте JavaScript Date (в подробностях рассмотренный в главе 2) и объекте, обозначающем текстовый диапазон (в IE для Windows он называется TextRange, а в W3C DOM — просто Range). Абстрактная природа этих объектов и сложность работы с ними могут привести ко множеству концептуальных затруднений.

Хотя в отдельных деталях реализации текстовых диапазонов в IE для Windows и W3C DOM несколько различаются, базовые операции одни и те же. В своей основе текстовый диапазон — это фрагмент текста, отделенного от HTML-элементов, окружающих этот текст или лежащих внутри него. Сам по себе текстовый диапазон не виден, но, если это необходимо для вашего приложения, его текст можно выделить.

У текстового диапазона есть начало и конец. Когда текстовый диапазон создается, положение его границ задается по умолчанию (опять же, в зависимости от типа DOM). Устанавливая их положения, можно пользоваться ссылками на HTML-элементы, но сам по себе текстовый диапазон не зависит от какой-либо иерархии, имеющейся в документе. При работе с диапазонами обычно сначала создается абстрактный диапазон, затем устанавливаются положения его границ, после чего выполняются все действия с текстом (например, его удаление или копирование). Например, можно преобразовать выделенный пользователем текст в элемент span, внешний вид которого будет контролироваться отдельным правилом CSS (см. рецепт 15.2).

Сравнивая различные реализации, можно сказать, что объект TextRange в IE более гибок и предоставляет больше возможностей для решения практических

задач. Только этот объект поддерживает средства для поиска текста на странице (и способ расположить границы диапазона по найденному тексту). Важно, что объект TextRange применим и к содержимому элементов textarea. Поэтому, хотя может показаться удобным использовать глобальные операции поиска и замены на странице (см. рецепт 15.3), полезнее задействовать их в элементе textarea, содержащем пользовательский текст. Конечно, браузеры IE 5.5 для Windows и старше поддерживают возможность сделать текст на странице редактируемым (хотя, чтобы изменения могли сохраниться, необходимо реализовать соответствующий серверный механизм).

Так как многие из рецептов этой главы связаны с позиционируемыми элементами, большинство из них используют библиотеку DHTML API из рецепта 13.3. Если сценарии зависят от функций этой библиотеки, это всегда оговаривается. Вы также можете создавать собственные версии этой библиотеки, включающие только нужные функции, хотя библиотека и не велика. Вы также можете переместить применяемые функции в другую библиотеку или даже в код на странице. Кроме того, рецепты, пользующиеся cookie для сохранения настроек, задействуют библиотеку cookies.js из рецепта 1.9.

15.1. Случайный афоризм на странице

NN4 IE4

Задача

Необходимо, чтобы в заданном месте страницы появилось известное высказывание, случайно выбранное из библиотеки.

Решение

Обратно-совместимый способ решения (до браузеров 4-й версии) опирается на методе document.write() для того, чтобы вставить текст в страницу в процессе загрузки. Но для начала в головной секции необходимо создать массив высказываний. Каждый элемент массива представляет собой объект, содержащий текст высказывания и имя автора:

```
var quotes = new Array();
quotes[quotes.length] = {quote:"One should eat to live, and not live to
                         eat.", author:"Moliere"};
quotes[quotes.length] = {quote:"For man is man and master of his fate."
                          author:"Tennyson"};
quotes[quotes.length] = {quote:"History is more or less bunk "
                          author :"Henry Ford"};
quotes[quotes length] - {quote:"You should never have your best trousers
                          on when you turn out to fight for freedom and
                          truth.", author:"Ibsen"};
quotes[quotes.length] = {quote:"It is vain and foolish to talk of knowing
                          Greek ", author:"Woolf"};
```

Следующая функция getSaying() использует случайные числа, чтобы получить индекс высказывания в массиве. Затем эта функция формирует HTML-код того, что должно появиться на странице:

```
function getSaying() {
    var currIndex = Math.floor(Math.random() * (quotes.length));
    var output = "<p class='quote'>" + quotes[currIndex].quote;
    output += "<span class='author'>   — " +
        quotes[currIndex] author;
    output += "</span></p>"
    return output:
}
```

Теперь, чтобы поместить на страницу случайное высказывание, следует включить в нее сценарий, вызывающий getSaying():

```
<script type="text/javascript">document.write(getSaying())</script>
```

Обсуждение

Суть решения — во взаимодействии между массивом объектов и методикой извлечения нужного объекта при необходимости. Формирующий массив код применяет для определения индексов новых записей текущую длину массива. Это упрощает редактирование списка высказываний, так как нет необходимости заново расставлять индексы каждый раз, как вы добавляете или убираете объект.

Формула для вычисления (псевдо) случайного числа здесь упрощена, так как искомые числа лежат в диапазоне между нулем и числом элементов массива. Полученное значение индекса используется для обращения к двум свойствам хранящегося в массиве объекта.

Оформление HTML полностью зависит от дизайна вашей страницы. В приведенном решении двум частям высказывания присвоены имена классов. Несложная таблица стилей, задающая отдельные характеристики шрифта для текста высказывания и имени автора, может выглядеть так:

```
<style  type-"text/css">
    .quote {font:bold 12px serif)
    .author {font:italic 12px serif)
</style>
```

Скорее всего, вы захотите использовать более подробные описания.

В приведенном решении после каждого обновления страницы показываются потенциально различные высказывания. Если вы хотите, чтобы одно и то же высказывание сохранялось в течение дня, то чтобы хранить дату и соответствующий ей индекс высказывания, следует добавить поддержку cookie. Обработку cookie можно поместить в функцию getSaying() так, чтобы она сначала проверяла значение cookie, хранящего информацию о дате. Если дата отличается от текущей, в cookie помещается ее текущее значение и вычисляется новое значение случайного индекса. Если дата совпадает с текущей, в качестве индекса высказывания используется сохраненное значение.

В Интернете встречается бесчисленное число приложений, использующих подобные ежедневные афоризмы. В некоторых из них определенная фраза или

фрагмент содержимого страницы связывается с днем недели или датой. Например, каждому дню недели в соответствие ставится его индекс (нумерация от 0 до 6, где воскресенью соответствует 0), имеющийся в объекте JavaScript Date. Поэтому применять номер дня недели в качестве индекса высказывания очень легко:

```
var today - new Date();
var currIdx - today.getDay();
```

Следует только убедиться, что в массивы имеется достаточное количество записей.

Еще одно приложение — вывод случайных изображений, например одного из изображений из каталога продукции. В этом случае массив объектов должен содержать информацию об изображении и URL:

```
var imgLinks = new Array();
imgLinks[imgLinks.length] - {src:"images/prods/x25.jpg",
    url="products/x25.html"};

function getRandomImage() {
    var currIndex = Math.floor(Math.random() * (quotes.length));
    var output = "<a href-'" + imgLinks[currIndex].url + "'>";
    output += "<img src='" + imgLinks[currIndex].src + "' alt='Product of
        the day!' />";
    output += "</a>"
    return output;
}
```

Благодаря этому при каждом посещении пользователь будет видеть изображение нового продукта со ссылкой, ведущей прямо на страницу этого продукта.

Смотрите также

Создание массивов смотрите в рецепте 3.1. Создание объектов — в рецепте 3.8.

15.2. Преобразование выделения в элемент

NN нет **IE4(Win)**

Задача

Необходимо преобразовать выделенный пользователем текст в элемент, чтобы выделить его цветом.

Решение

Начнем с контейнера абзаца, который отслеживает выделение текста путем обработки события onmouseup:

```
<p onmouseup="selection2Element()">Lorem ipsum dolor sit amet, consectetaur
adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. </p>
```

Обработчиком события здесь служит функция selection2Element(), создающая поддерживаемый IE объект TextRange, границы которого совпадают с границами выделения в тот момент, когда пользователь отпускает кнопку мыши:

```
function selection2Element() {
    var rng = document.selection.createRange();
    var newHTML = "<span class='newSpan'>" + rng.text + "</span>";
    rng.pasteHTML(newHTML);
}
```

В этом примере предполагается, что в документе заранее задано описание стиля для класса newSpan. Имеющийся у диапазона метод pasteHTML() позволяет заменить выделенный текст на HTML-код, в результате чего тексту присваивается определенное значение стиля. Все, что видит в результате этих действий пользователь, — изменение стиля выделенного текста.

Обсуждение

Предложенное решение следует применять с осторожностью, так как оно легко может вызвать серьезные проблемы. Если выделение пересечет границу HTML-контейнера, новый HTML-код, которым функция selection2Element() заменит существующий, будет как минимум некорректным, так как контейнеры будут в нем перекрываться. В худшем случае замена приведет к разрушению структуры HTML-кода по соседству.

Один из способов предотвратить подобные проблемы основан на использовании поддерживаемых только IE обработчиков событий onselectstart помимо onmouseup. Обработчик onselectstart следует применять для того, чтобы сохранить в глобальной переменой ссылку на последний элемент, в котором началось выделение текста (получить эту ссылку можно из свойства event.srcElement). Затем в функции selection2Element() нужно сохранить новое значение event.srcElement с сохраненным, и если ссылки совпадают, выделение целиком находится внутри одного элемента:

```
// глобальная переменная
var selectionStart,

// обработчик onselectstart
function saveStart() {
    selectionStart = event.srcElement;
}
// обработчик onmouseup
function selection2Element() {
    if (event.srcElement == selectionStart) {
        var rng = document.selection.createRange();
        var newHTML = "<span class='newSpan'>" + rng.text + "</span>";
        rng.pasteHTML(newHTML);
    } else {
        alert("Выделение не должно выходить за пределы абзаца.");
    }
```

В дальнейшем можно еще больше упростить применение этого рецепта, переместив обработчики событий на верхний уровень документа, а абзацам, содер-

жащим выделяемый текст, назначить опереженное имя класса. Затем следует ограничить выполнение функции selection2Element(), обрабатывая только те элементы, которые относятся к нужному классу.

Имеющийся в W3C DOM объект Range, в том виде, в котором он реализован в браузерах Netscape, не имеет достаточных возможностей для реализации этого решения.

Смотрите также

Более сложное приложение объекта TextRange в IE рассматривается в рецепте 15.3.

15.3. Программирование поиска и замены в тексте документа

NN нет **IE4(Win)**

Задача

Необходимо реализовать поиск и замену текста в документе.

Решение

Так как этот рецепт во многом зависит от особенностей объекта TextRange в IE, (не реализованных в IE для Mac вплоть до версий 5.x), он работает только в Internet Explorer 4 и старше для Windows.

Для автоматизации операций поиска и замены с помощью объекта TextRange предлагается использовать приведенную в листинге 15.1 библиотеку rangeReplace.js. В зависимости от типа операции и опыта пользователя можно выбрать одну из двух функций. Для многократной замены без предупреждения перед каждой операцией следует вызывать функцию srBatch():

```
srBatch(document.body, "Law", "legislation" false, false):
```

Пять аргументов этой функции означают соответственно:

- ссылка на элемент, содержащий текст;
- строка для поиска;
- строка для замены;
- учитывать ли регистр символов (булевское значение);
- производить ли поиск и замену целых слов (булевское значение).

Аналогичная функция srQuery() требует тот же набор аргументов, но перед каждой заменой выделяет текст и запрашивает пользователя:

```
srQuery(document.body, "Law", "legislation" false, false):
```

Чтобы отменить последние изменения, сделанные одной из функций, используйте функцию undoReplace(), не требующую аргументов.

Обсуждение

К любому документу, в котором необходима описанная выше функциональность, следует присоединить библиотеку rangeReplase.js, приведенную в листинге 15.1.

Листинг 15.1. Библиотека rangeReplace.js

```
// Глобальная переменная, содержащая объект диапазона
var rng:

// Формирует третий аргумент TextRange.findText()
function getArgs(caseSensitive. wholeWord) {
    var isCaseSensitive = (caseSensitive) ? 4 : 0:
    var isWholeWord - (wholeWord) ? 2 : 0:
    return isCaseSensitive * isWholeWord:
}

// Поиск и замена без запроса
function srBatch(container, search, replace. caseSensitive. wholeWord) {
    if (search) {
        var args = getArgs(caseSensitive, wholeWord):
        rng - document.body.createTextRange();
        rng.moveToElementText(container);
        clearUndoBuffer();
        for (var i = 0: rng.findText(search, 1000000. args): i++) {
            rng.text = replace:
            pushUndoNew(rng, search, replace):
            rng.collapse(false)
        }



// Поиск и замена с подтверждением каждой операции
function srQuery(contai.er. search. replace. caseSensitive. wholeWord) {
    if (search) {
        var args - getArgs(caseSensitive. wholeWord):
        rng = document. body. createTextRangeO:
        rng.moveToElementText(container);
        clearUndoBufferO:
        while (rng.findText(search, 10000. args)) {
            rng.selectO:
            rng. scrollIntoViewO;
            if (confirm("Replace?")) {
                rng.text = replace:
                pushUndoNew(rng. search. replace):
            }
            rng collapse(false)  .
        ]
    }
}


    Буфер отмены

// Временное хранилище информации для отмены
var undoObject = {origSearchString:"" newRanges .[]}:
```

```
// Сохранение исходной строки для поиска и информации о каждой произведенной замене
function pushUndoNew(rng. srchString. replString) {
    undoObject.origSearchString - srchString:
    rng.moveStart("character". -replString.length);
    undoObject.newRanges[undoObject.newRanges.length] = rng.getBookmark();
}

// Очистка массива и глобальной переменной, хранящей строку для поиска
function clearUndoBuffer() {
    undoObject.origSearchString = "":
    undoObject.newRanges.length = 0:
}

// Выполнение отмены
function undoReplace() {
    if (undoObject.newRanges.length && undoObject.origSearchString) {
        for (var i = 0: i < undoObject.newRanges.length; i++) {
            rng.moveToBookmark(undoObject.newRanges[i]);
            rng.text - undoObject.origSearchString;
        }
        clearUndoBuffer();
    }
}
```

В начале библиотеки расположена вспомогательная функция, getArgs(), используемая обеими функциями поиска и замены. Имеющийся у объекта TextRange в IE метод findText() требует три аргумента. Первый — это строка для поиска, затем указывается количество символов, в пределах которых производится поиск, а третий аргумент содержит код с дополнительной информацией о поиске. Возможны четыре значения этого кода: 0 (поиск частей слов), 1 (поиск в обратном направлении), 2 (поиск целых слов) или 4 (учитывать регистр символов). С помощью операций двоичной арифметики нужные коды следует объединить в одно значение. Функция getArgs() преобразует два передаваемых ей булевских признака в соответствующий код, нужный для findText().

Обе функции для поиска и замены, srBatch() и srQuery(), применяют одни и те же методы для создания диапазона, задания его границ, очистки буфера отмены и выполнения поиска и замены. Разница в том, что srQuery() выделяет каждый найденный фрагмент текста и посылает запрос, нужно ли его заменять.

Обе функции снабжены одинаковыми возможностями для отмены изменений. Все сделанные в ходе операции поиска и замены изменения сохраняются в глобальный объект undoObject с помощью закладок объекта TextRange (они позволяют сохранять описание диапазона, чтобы применить его позднее). При каждой новой замене функция pushUndoNew() сохраняет информацию об операции в свойствах объекта undoObject. Сохранять положение диапазона следует уже после того, как произведена замена, чтобы впоследствии вставленный текст можно было бы удалить и заменить на исходный.

Прежде чем начинать поиск и замену в документе, сохраненные в undoBuffer значения должны быть удалены с помощью функции clearUndoBufferO, чтобы не возникало ненужного взаимодействия с предыдущей операцией. Когда пользователь захочет отменить сделанные изменения, undoReplace() переберет все сохраненные в undoObject диапазоны, заменяя текст в них на исходное значение.

У любой из двух предложенных функций поиска и замены первым аргументом является ссылка на HTML-контейнер, содержащий обрабатываемый текст. Хотя вы можете передать в функцию ссылку на весь документ целиком, выполнение функции в этом случае может привести к изменению текста на подписях в формах и других местах, где это не нужно. Таким образом, HTML-документ следует структурировать так, чтобы подвергаемый обработке текст был отделен от неизменяемого. Например, набор абзацев всегда можно обернуть в элемент span, чтобы получить нужный контейнер.

Следует различать закладки объекта TextRange и закладки, имеющиеся в браузере (меню Избранное). Возвращаемое методом rangeObject.getBookmark() значение закладки представляет собой содержащую непонятный для человеческого глаза текст строку, которую браузер может интерпретировать как однозначное описание диапазона. Передав значение закладки методу rangeObject.moveToBookmark(), можно восстановить положения границ диапазона. Можно сохранять сколь угодно много подобных закладок.

Объект Range, имеющийся в W3C DOM, хотя и имеет некоторые характеристики объекта TextRange в IE, недостаточно гибок для решения задачи поиска и замены. Возможно, со временем стандарт и браузер будут усовершенствованы и станут пригодными для выполнения операций поиска и замены.

Смотрите также

Информация о текстовых диапазонах имеется во вступлении к этой главе.

15.4. Создание слайд-шоу

NN6 **IE5.5**

Задача

Необходимо создать в браузере эквивалент презентации PowerPoint.

Решение

Существует множество способов имитации слайд-шоу, и в этом рецепте предлагается простой, удобный в применении в современных браузерах способ. Все содержимое, таблицы стилей и сценарии размещены внутри единственной HTML-страницы, код которой приведен в листинге 15.2 в обсуждении.

Чтобы переделать этот пример под свои нужды, содержимое каждого отдельного слайда следует поместить внутрь отдельного элемента div, у которого атрибуту class должно быть присвоено значение slide. Отдельные слайды следует расположить в документе в том же порядке, в котором они должны демонстрироваться.

Для изменения внешнего вида слайдов можно воспользоваться таблицами стилей. Оформление показанной в листинге страницы соответствует обычному для презентационных программ оформлению слайда.

Обсуждение

В листинге 15.2 приведена HTML-страница, служащая для демонстрации слайд-шоу (на ней, чтобы сэкономить место, сделан только один слайд). Благодаря тому, что все слайды находятся на одной странице, не возникает задержки при переходе между ними.

Листинг 15.2. HTML-страница и сценарии для показа слайдов

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
<head>
<title>DHTML Slide Show</title>
<style type="text/css">
#slides {font-family:Verdana, Arial, sans-serif;
         position:absolute;
         top:40px;
         width:90%
        }
.slide {position:absolute;
        top:0px;
        left:0px;
        display:none;
        width:80%;
        height:500px;
        overflow:hidden;
        background-color:#ccffcc;
        font-size:18px;
        padding:20px;
        border:5px solid #ff9900;
        margin:10%
       }
h1 {text-align:right;
    padding-right:10%
   }
h2 {font-size:36px;
    text-align:center
   }
li {list-style-image:url(end.gif);
    list-style-position:outside
   }
hr {width:60%;
    height:5px;
    background-color:#ff9900
   }
#titleBar {width:100%;
           height:10px
          }
body {background-color:#339966}
#controller {position:absolute;
             top:30px;
             left:10%
```

продолжение ↓

Листинг 15.2 (продолжение)

```
</style>
<script type="text/javascript" src="../js/DHTMLAPI.js"></script>
<script type="text/javascript">
// Массив всех слайдов
var allSlides;
// Счетчик слайдов
var currSlide = -1;

// Формирование глобального массива слайдов
function getAllSlides() {
    var allChildren = document.getElementById("slides").childNodes;
    var slideElems = new Array();
    for (var i = 0; i < allChildren.length; i++) {
        if (allChildren[i].nodeType == 1) {
            slideElems[slideElems.length] = allChildren[i];
        }
    }
    allSlides = slideElems;
}

// Задание высоты каждого слайда на все окно
function setHeights() {
    for (var i = 0; i < allSlides.length; i++) {
        allSlides[i].style.height = getInsideWindowHeight() - 200 + "px"
    }
}

// Переход к следующему слайду
function next() {
    if (currSlide < 0) {
        allSlides[++currSlide].style.display = "block";
    } else if (currSlide < allSlides.length - 1) {
        allSlides[currSlide].style.display = "none";
        allSlides[++currSlide].style.display = "block";
    } else if (currSlide = allSlides.length - 1) {
        allSlides[currSlide++].style.display = "none";
    }
}

// Переход к предыдущему слайду
function prev() {
    if (currSlide > allSlides.length - 1) {
        allSlides[--currSlide].style.display = "block"
    } else if (currSlide > 0) {
        allSlides[currSlide].style.display = "none";
        allSlides[--currSlide].style.display = "block"
    } else if (currSlide == 0) {
        allSlides[currSlide--].style.display = "none";
    }
}

// Инициализация показа слайдов
function initSlides() {
```

```
    getAllSlides();
    setHeights();
}
</script>
</head>
<body onload="initDHTMLAPI(); initSlides()" onresize="setHeights()">
<h1>Билль о правах Соединенных Штатов</h1>
<hr id="titleBar" />
<div id="controner">
<form>
<input type="button" value="Предыдущий" onclick="prev()" />
<input type="button" value="Следующий" onclick="next()" />
</form>
</div>

<div id="slides">

<div id="slide1" class="slide">
<h2>ARTICLE I</h2>
<hr />
<p>
Congress shall make no law respecting an establishment of religion, or prohibiting the free
exercise thereof: or abridging the freedo~ of speech, or of the press: or the right of the
people peaceably to assemble, and to petition the government for a redress of grievances.
</p>
<ul>
<li>Note 1</li>
<li>Note 2</li>
<li>Note 3</li>
</ul>
</div>

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

Хотя показанный выше код работает в IE 5 и старше, а также в NN 6 и старше для нормальной обработки CSS в этом примере требуется IE 5.5 или старше для Windows. Внешний вид слайда приведен на рис. 15.1

Внешний вид отдельных слайдов зависит по большей части от правил CSS примененных к последовательности элементов div (с классом slide), вложенных во внешний, абсолютно позиционируемый контейнер div (с идентификатором slides). Дополнительные правила таблицы стилей контролируют вид элементов внутри слайда, таких как заголовки, маркированные списки и горизонтальны разделители.

HTML-код заголовочной части слайда, с кнопками управления и содержимым имеет простую последовательную структуру. Кнопки управления просто вызыва ют функции next() и prev() для перехода к соответствующему слайду. С помощью CSS-правил кнопки управления располагаются в свободной части страницы Инициализацию как DHTML API, так и сценариев для показа слайдов вызыва ет обработчик события onload тела документа.

Листинг 15.2 (продолжение)

```
</style>
<script type="text/javascript" src="../js/DHTMLAPI.js"></script>
<script type="text/javascript">
// Массив всех слайдов
var allSlides;
// Счетчик слайдов
var currSlide = -1;

// Формирование глобального массива слайдов
function getAllSlides() {
    var allChildren = document.getElementById("slides").childNodes;
    var slideElems = new Array();
    for (var i = 0; i < allChildren.length; i++) {
        if (allChildren[i].nodeType == 1) {
            slideElems[slideElems.length] = allChildren[i];
        }
    }
    allSlides = slideElems;
}

// Задание высоты каждого слайда на все окно
function setHeights() {
    for (var i = 0; i < allSlides.length; i++) {
        allSlides[i].style.height = getInsideWindowHeight() - 200 + "px"
    }
}

// Переход к следующему слайду
function next() {
    if (currSlide < 0) {
        allSlides[++currSlide].style.display = "block"
    } else if (currSlide < allSlides.length - 1) {
        allSlides[currSlide].style.display = "none";
        allSlides[++currSlide].style.display = "block"
    } else if (currSlide == allSlides.length - 1) {
        allSlides[currSlide++].style.display = "none";
    }
}

// Переход к предыдущему слайду
function prev() {
    if (currSlide > allSlides.length - 1) {
        allSlides[--currSlide].style.display = "block";
    } else if (currSlide > 0) {
        allSlides[currSlide].style.display = "none";
        allSlides[--currSlide].style.display = "block";
    } else if (currSlide == 0) {
        allSlides[currSlide--].style.display = "none";
    }
}

// Инициализация показа слайдов
function initSlides() {
```

```
    getAllSlides();
    setHeights();
}
</script>
</head>
<body onload="initDHTMLAPI(); initSlides()" onresize="setHeights()">
<h1>Билль о правах Соединенных Штатов</h1>
<hr id="titleBar" />
<div id="controller">
<form>
<input type="button" value "Предыдущий" onclick="prev()" />
<input type="button" value="Следующий" onclick="next()" />
</form>
</div>

<div id="slides">

<div id="slide1" class="slide">
<h2>ARTICLE I</h2>
<hr />
<p>
Congress shall make no law respecting an establishment of religion, or prohibiting the free
exercise thereof: or abridging the freedom of speech, or of the press, or the right of the
people peaceably to assemble, and to petition the government for a redress of grievances.
</p>
<ul>
<li>Note 1</li>
<li>Note 2</li>
<li>Note 3</li>
</ul>
</div>

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

Хотя показанный выше код работает в IE 5 и старше, а также в NN 6 и старше, для нормальной обработки CSS в этом примере требуется IE 5.5 или старше для Windows. Внешний вид слайда приведен на рис. 15.1

Внешний вид отдельных слайдов зависит по большей части от правил CSS, примененных к последовательности элементов **div** (с классом slide), вложенных во внешний, абсолютно позиционируемый контейнер div (с идентификатором slides). Дополнительные правила таблицы стилей контролируют вид элементов внутри слайда, таких как заголовки, маркированные списки и горизонтальные разделители.

HTML-код заголовочной части слайда, с кнопками управления и содержимым имеет простую последовательную структуру. Кнопки управления просто вызывают функции next() и prev() для перехода к соответствующему слайду. С помощью CSS-правил кнопки управления располагаются в свободной части страницы. Инициализацию как DHTML API, так и сценариев для показа слайдов вызывает обработчик события onload тела документа.

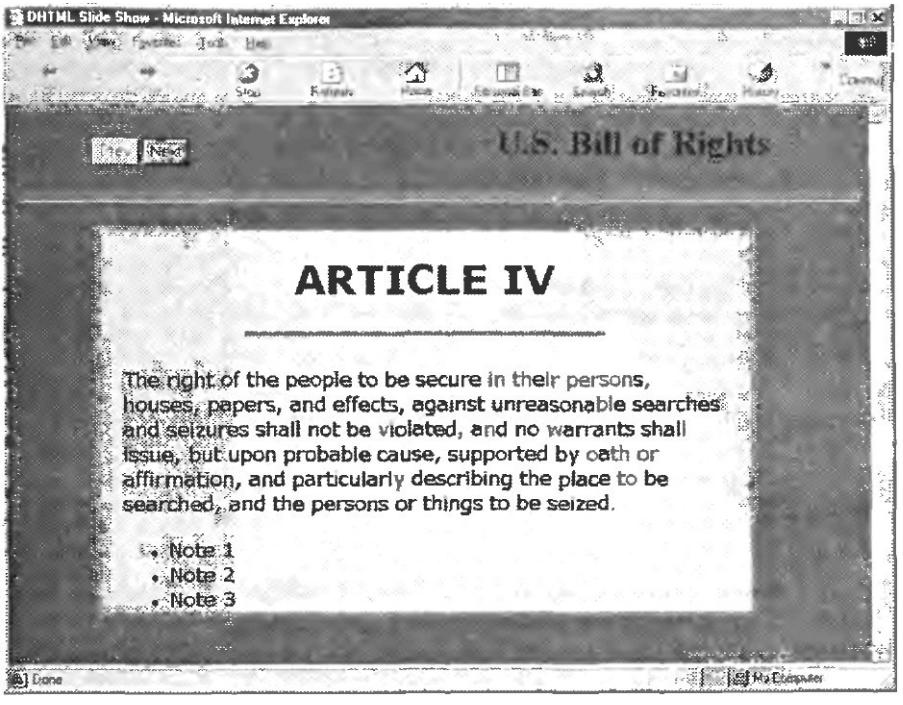


Рис. 15.1. Окно со слайдом

Показ слайдов производится под управлением сценария, который скрывает и отображает отдельные элементы div, представляющие слайды. Таким образом, сценарий просто должен отслеживать, какой слайд демонстрируется в данный момент, и показывать следующий или предыдущий. Вместо того чтобы использовать для описания последовательности слайдов жестко заданные идентификаторы, в этом рецепте сценарий определяет последовательность узлов в дереве документа. Каждый слайд является дочерним узлом внешнего контейнера (идентификатор которого slides). Функция getAllSlides() сохраняет ссылки на все слайды в глобальную переменную allSlides в том порядке, в котором они расположены на странице. Следует учитывать, что эта функция вызывается обработчиком события onload, так как слайды должны полностью загрузиться перед тем, как их можно будет пересчитать.

Счетчиком слайдов, указывающим на то, какой из них сейчас отображается, служит отсчитываемый от нуля индекс. Изначально глобальная переменная currSlide инициализируется значением -1, чтобы указать, что не показывается ни одного слайда и что следующим следует показать первый слайд (с индексом 0). Как функция prev(), так и next() применяют значение переменной currSlide для того, чтобы определить индекс следующего или предыдущего слайда в последовательности, до тех пор пока не достигнуто одно из двух граничных условий, когда пользователь достигает начала или конца показа. Обе эти функции должны проверять эти граничные условия и в соответствии с ними управлять переменной currSlide.

Наконец, осталось сделать так, чтобы размер области слайда не менялся при переходе от одного слайда к другому (в зависимости от его высоты), для чего следует фиксировать высоту области. Но для того, чтобы высота была пропорциональна высоте окна браузера, нужна помощь DHTML. Функция setHeights() настраивает высоту всех слайдов, используя библиотеку DHTML API. Обратите внимание на то, что setHeights() вызывается не только из обработчика onload, но и из onresize элемента body. Это позволяет поддерживать правильную высоту слайда во всех случаях.

Внешний вид и поведение основанных на DHTML презентаций имеет столько же вариаций, сколько существует веб-дизайнеров. Большая часть таблиц стилей в этом рецепте определяет только размещение элементов. Размеры, шрифты и цвета можно выбирать по вашему усмотрению. То же самое касается и элементов управления, реализованных здесь как простые кнопки на HTML-форме. Можно применять вместо них изображения или же вообще не использовать специальные элементы управления, переходя между слайдами просто по щелчку в любом месте окна. Добавив к показанному рецепту следующий код, можно получить подобную функциональность (за исключением IE для Mac). Этот код также предоставляет способ переходить в обратном направлении, удерживая при щелчке на окне Shift:

```
function changeSlide(evt) {
    evt = (evt) ? evt : window.event;
    if (evt.shiftKey) {
        prev();
    } else {
        next();
    }
}
```

Одной из сторон типичной демонстрации слайдов, не отраженной в предыдущем коде, являются эффекты перехода. Можно реализовать множество совместимых с разными браузерами эффектов, например, когда старый слайд выдвигается за левую грань экрана, в то время как новый появляется справа. Но скорость и плавность анимации такого рода во многом зависят от браузера и рабочего окружения. В то же время благодаря тесной интеграции Internet Explorer с операционной системой Windows этот браузер предоставляет большое количество эффектов перехода, задаваемых как правила таблиц стилей.

Список поддерживаемых эффектов довольно большой, как и количество настроек, поддерживаемых отдельными эффектами (см. рецепт 12.9). В качестве примера можно рассмотреть следующее CSS-правило, которое следует добавить к списку правил для класса .slide:

```
filter:progid:DXImageTransform.Microsoft.Iris(IrisStyle="circle")
```

Затем необходимо изменить функции next() и prev() так, чтобы они применяли эффект перехода, когда он будет нужен. К примеру, функция next() должна быть переделана так:

```
function nextO {
    var allSlides = getAllSlides();
    if (currSlide < 0 ) {
```

```
        allSlides[++currSlide].filters[0].apply();
        allSlides[currSlide].style.visibility = "visible";
        allSlides[currSlide].filters[0].play();
    } else if (currSlide < allSlides.length ) {
        all Slides[currSlide].style.visibility= "hidden";
        allSlides[++currSlide].filters[0].apply();
        allSlides[currSlide] style visibility = "visible";
        allSlides[currSlide].filters[0].play();
    } else if (currSlide == allSlides.length - 1 ) {
        allSlides[currSlide++].style.visibility = "hidden";
    }
}
```

Чтобы эффекты перехода работали в IE для Windows и не приводили к ошибкам сценария в IE для Mac или в браузерах Netscape, следует еще немного увеличить код:

```
function next() {
    var allSlides = getAllSlides();
    var nextSlide;
    if (currSlide < 0 ) {
        nextSlide = allSlides[++currSlide];
        if (nextSlide.filters) {
            nextSlide.filters[0].apply();
        } else {

        nextSlide.style.visibility = "visible";
        if (nextSlide.filters) {
            nextSlide.filters[0].play();
        }
    } else if (currSlide < allSl ides.length ) {
        allSlides[currSlide].style.visibility = "hidden";
        nextSlide = allSlides[++currSlide];
        if (nextSlide.filters) {
            nextSl ide. f i lters[0].apply();
        }
        allSlides[currSlide].style.visibility - "visible";
        if (nextSlide.filters) {
            nextSlide.filters[0].play();
        }
    } else if (currSlide == allSlides.length - 1 ) {
        allSlides[currSlide++].style.visibility = "hidden";
    }
}
```

Наконец, можно сделать так, чтобы слайды сменялись автоматически, предоставляя пользователю некоторое время на просмотр. Функцию next() можно переделать так, чтобы она автоматически переходила в начало после завершающего слайда (для этого не нужна функция prev())

```
function next 0 {
    var allSlides = getAllSlides();
    if (currSlide < 0) {
        allSlides[++currSlide].style.display = "block";
```

```
    } else if (currSlide < allSlides.length - 1) {
        allSlides[currSlide].style.display = "none":
        allSlides[++currSlide].style.display = "block":
    } else if (currSlide ==allSlides.length - 1) {
        allSlides[currSlide++].style.display = "none";
        currSlide - -1:
        next();
    }
}
```

Затем в обработчике onload следует использовать **setInterval()** для того, чтобы вызывать функцию next() каждые несколько секунд. Так в показанном выше примере обработчиком являлась функция setHeights(), вызов **setInterval()** можно добавить в ее конец. Но так как эта функция вызывается в тех случаях, когда изменяется размер страницы, перед тем как установить новый таймер, следует отключить старый. Если этого не сделать, после изменения размера окна будет запущено два таймера, и слайды будут сменяться вдвое быстрее. Идентификатор запущенного таймера необходимо сохранять в глобальной переменной:

```
Var slideInterval;

function setHeights() {
    clearInterval(slideInterval);
    var allSlides = getAllSlides();
    for (var i = 0; i < allSlides.length; i++) {
        allSlides[i].style.height = getInsideWindowHeight() - 200 + "px".
    }
    slideInterval = setInterval("next()", 5000);
}
```

Смотрите также

Как поместить на страницу позиционируемый элемент, рассказывается в рецепте 13.1. Подробности о библиотеке DHTML API смотрите в рецепте 13.3. Введение в эффекты перехода в Internet Explorer для Windows смотрите в рецепте 12.9.

15.5. Автоматическая прокрутка страницы

NN4 **IE4**

Задача

Необходимо сделать так, чтобы страница автоматически прокручивалась по вертикали.

Решение

Начиная с четвертой версии, объект window в браузерах Internet Explorer и Netscape был снабжен методом **scrollBy()**. Постоянно вызывая его с помощью

setInterval(), можно прокручивать окно браузера без помощи пользопателя Вот пример того, как это можно сделать:

```
function scrollWindow() {
    window.scrollBy(0. 1):
}
function initScroll() {
    setInterval("scrollWindow()' 100),
}
```

Для того чтобы запустить автоматическую прокрутку, следует вызвать функцию initScroll(), например, из обработчика события onload.

Обсуждение

Определить оптимальную скорость прокрутки непросто, что может побудить вас отказаться от применения этого рецепта. Скорость прокрутки управляется двумя переменными: количеством пикселов, на которое каждый раз производится прокрутка (второй параметр функции scrollBy()), и тем, насколько часто вызывается scrollWindow() (интервал, передаваемый функции setInterval()). Если приращение чересчур велико, страница прокручивается рывками, из-за чего за ней становится трудно уследить. Поэтому чем меньше величина приращения, тем лучше, что и показано в рецепте, использующем величину приращения в один пиксел.

Номинально частота задается числом миллисекунд между последовательными вызовами функции scrollWindow(). Если это число слишком велико, страница прокручивается слишком медленно даже для посетителей, читающих со средней скоростью. В идеальной ситуации наиболее удобной для пользователей была бы небольшая задержка, но при этом возникает другая проблема: дело в том, что при использовании setInterval() промежутки времени между последовательными вызовами не совсем одинаковы. Поэтому чем меньше величина интервала, тем заметней становятся рывки и задержки на фоне плавной прокрутки.

Чтобы определить, какая часть страницы уже прокручена, можно считывать мгновенное значение величины прокрутки (в пикселах). Тем не менее синтаксис различается не только в IE и Netscape, но и в разных режимах совместимости IE 6. В Netscape 6 и старше используется доступное только для чтения свойство window.scrollX и window.scrollY. Но в IE старых версий и в IE 6 в режиме обратной совместимости следует применять свойства document. body.scrollLeft и document.body.scrollTop. В IE 6 в режиме CSS-совместимости применяются свойства document.body.parentElement.scrollLeft и document.body.parentElement.scrollTop. Последние два свойства показывают величину прокрутки всего элемента html, заполняющего окно.

Смотрите также

Альтернативный способ реализации прокрутки содержимого без прокрутки окна (в подвижном контейнере div) рассматривается в рецепте 13 12.

15.6. Приветствие с учетом времени суток

NN2 **IE3**

Задача

Следует поместить на страницу приветствие, учитывающее время суток у пользователя, например «Добрый день» или «Добрый вечер».

Решение

Для начала необходимо написать функцию, возвращающую строки, ассоциированные с каждым временем суток, которое определяется с помощью объекта Date:

```
function dayPart() {
    var oneDate = new Date();
    var theHour = oneDate.getHours()
    if (theHour < 12) {
        return "утра";
    } else if (theHour < 18) {
        return "дня";
    } else {
        return "вечера";
    }
}
```

Учитывая возможность того, что браузер может не поддерживать сценарии, код сценария следует обрамить HTML-тегами комментария и добавить элемент <noscript>, содержащий текст для таких браузеров:

```
<script languag = JavaScript" type="text/javascript">
<!-
document.write("Доброго вам " + dayPartO + " и добро пожаловать")
//->
</script>
<noscript>Добро пожаловать</noscript>
 на сайт компании GiantCo.
```

Обсуждение

После того как страница загружена, создается объект Date. Так как в его конструктор не передаются параметры, для генерации объекта используются текущие значения даты и времени. Следует учитывать, что объект Date — не часы, отсчитывающие время, а скорее мгновенный снимок часов в момент его создания. Точность представления времени зависит только от точности внутренних часов на компьютере пользователя.

У объекта Date есть множество методов для считывания и установки значения отдельных компонентов, начиная с миллисекунд и заканчивая годами. В решении применяется метод getHours(), возвращающий целое число в пределах от 0 до 23, соответствующее времени на часах пользователя. Функция dayPart() просто делит день на три части, разграниченные полуднем и 18 часами и соответствующие утру, дню и вечеру.

Смотрите также

Дополнительная информация о работе с объектом Date имеется в рецептах с 2.9 по 2.11.

15.7. Отображение времени до Рождества

NN4 IE4

Задача

Следует отобразить на странице время, оставшееся до наступления некоторой даты (или времени) в локальном часовом поясе пользователя.

Решение

Ниже приведена функция, возвращающая число дней, оставшихся до Рождества, в часовом поясе пользователя:

```
function getDaysBeforeNextXmas() {
    var oneMinute = 60 * 1000;
    var oneHour = oneMinute * 60;
    var oneDay = oneHour * 24;
    var today = new Date();
    var nextXmas = new Date();
    nextXmas.setMonth(11);
    nextXmas.setDate(25);
    if (today.getMonth() == 11 && today.getDate() > 25) {
        nextXmas.setFullYear(nextXmas.getFullYear() + 1);
    }
    var diff = nextXmas.getTime() - today.getTime();
    diff = Math.floor(diff/oneDay);
    return diff;
}
```

Возвращаемое значение представляет собой целое число, равное числу дней. Это число можно поместить на страницу в процессе загрузки или после завершения загрузки:

```
<p>До Рождества осталось всего
<script type="text/javascript">document.write(getDaysBeforeNextXmas())</script>
 дней.</p>
```

Обсуждение

Решение задачи об определении числа дней, оставшихся до конкретной даты, связано с некоторыми побочными последствиями, которые следует учесть, прежде чем применять решение. Если рассматривать приведенный в решении случай, функция может работать год за годом без дополнительной поддержки, так как дата не привязана к какому-то определенному году. Отдельное условие if необходимо для

учета той ситуации, когда текущее время попадает в интервал между Рождеством и 1 января следующего года. Чтобы в результате вычислений было получено корректное значение, необходимо рассматривать дату в следующем году.

Можно переработать функцию, сделав ее многоцелевой. Для этого она должна получать информацию о целевой дате в будущем, в том числе и год. Такая функция может выглядеть так:

```
function getDaysBefore(year, month, date) {
    var oneMinute = 60 * 1000;
    var oneHour = oneMinute * 60;
    var oneDay = oneHour * 24;
    var today = new Date();
    var targetDate = new Date();
    targetDate.setYear(year);
    targetDate.setMonth(month);
    targetDate.setDate(date);
    alert(targetDate);
    var diff = targetDate.getTime() - today.getTime();
    diff = Math.floor(diff/oneDay);
    return diff;
}
```

Те, кто хорошо знакомы с объектом Date, могут поинтересоваться, почему при создании объекта Date, соответствующего целевому времени, аргументы не передаются напрямую в конструктор. Дело в том, что значение Date включает в себя минуты и часы. Сравнивая значения текущей даты и даты в будущем, важно сравнивать их в одно и то же время. Если же передать аргументы напрямую в конструктор, созданный объект будет соответствовать времени 00:00:00 соответствующей даты.

Хотя в этом примере показано только, как вычислить число дней, можно определить и другие единицы времени, изменив в предпоследней строке функции делитель с oneDay (это значение служит для определения числа дней) на oneHour (чтобы найти число часов) или oneMinute. Вычисления с датами производятся с точностью до миллисекунд, поэтому при необходимости можно рассматривать и меньшие интервалы времени.

Не обязательно отображать результат вычислений в виде текста на странице. Если есть графические изображения чисел от 0 до 9, можно преобразовать число в графическое изображение, составленное из этих картинок. Например, если картинки называются 0.jpg, 1.jpg и т. д., для формирования HTML-кода последовательности изображения можно применять следующую функцию:

```
function getDaysArt() {
    var output = "";
    var dayCount = getDaysBeforeNextXmas() + "";
    for (var i = 0; i < dayCount.length; i++) {
        output += "<img src='digits/" + dayCount.charAt(i) + ".jpg'";
        output += "height='120' width='60' alt='" + dayCount.charAt(i) +
            "' />";
    }
    return output;
}
```

Чтобы поместить этот код на страницу при загрузке, используйте метод document.write() таким образом:

```
<p>До Рождества осталось всего
<script type="text/javascript">document.write(getDaysArt())</script>
дней.</p>
```

Смотрите также

В рецепте 15.8. демонстрируется таймер, отсчитывающий время до определенной даты в определенном часовом поясе. В рецепте 15.9 показано, как сделать элемент управления для задания даты на основе календаря.

15.8. Таймер

NN4 **IE4**

Задача

Необходимо разместить на странице таймер, отсчитывающий время до заданного момента.

Решение

Подобные таймеры могут принимать разную форму. Пример решения может дать код в листинге 15.3, приведенный в обсуждении. В этом примере в качестве точки отсчета для таймера задается наступление нового года в часовом поясе пользователя, а на странице постоянно обновляется количество дней, часов, минут и секунд до наступления этого времени.

Для этого обработчик события onload вызывает функцию countDown() на странице с помощью метода setInterval():

```
<body onload="setInterval('countDown()', 1000)">
```

Надпись на странице обновляется примерно каждую секунду, так как метод setInterval() будет вызывать countDown() до тех пор, пока таймер не будет остановлен.

Обсуждение

Таймер можно оформить как текстовое поле ввода (в любом поддерживающем сценарии браузере), набор изображений (в IE 4, NN 3 и старше), текст в теле документа (в IE 4, NN 6 и старше). В этом примере применяются изображения, так как этот вариант дает наибольший выбор вариантов оформления. Но по некоторым синтаксическим причинам для этого рецепта подходит Navigator версии не ниже 4. Пример таймера показан на рис. 15.2.

Полный HTML-код этого примера, включая сценарии, приведен в листинге 15.3.

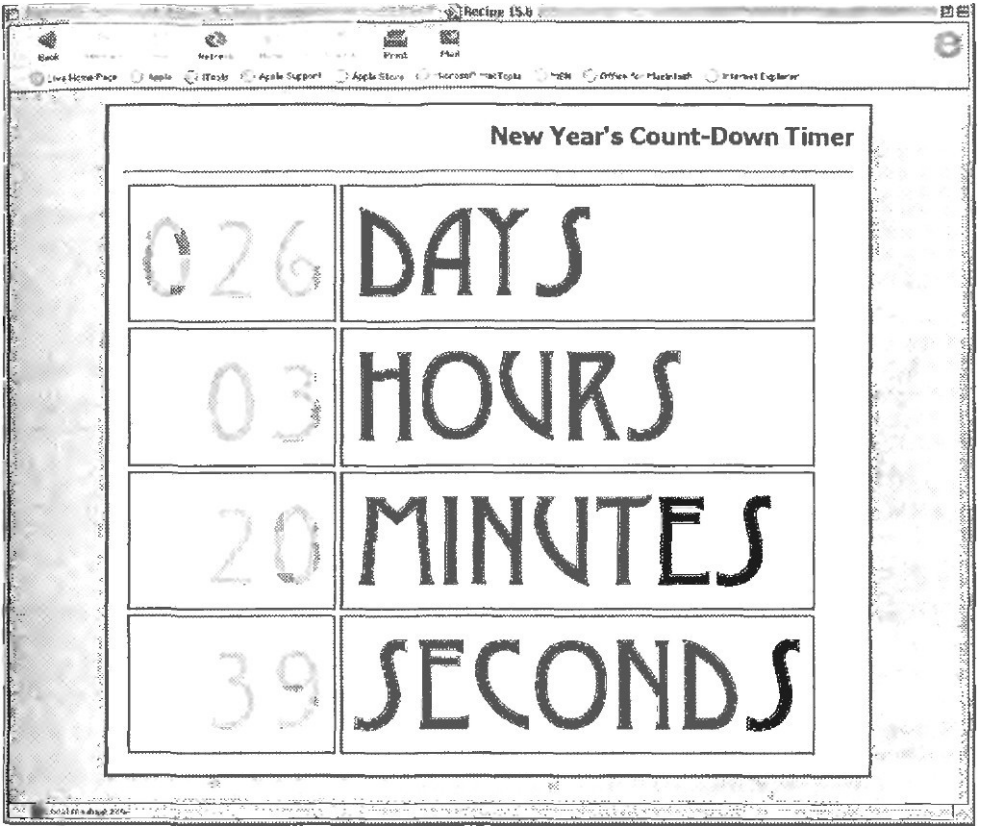


Рис. 15.2. Таймер, оформленный с помощью изображений

Листинг 15.3. Страница с таймером

```
<html>
<head>
<title>Recipe 15.8</title>
<link rel="stylesheet" id="mainStyle" href="../css/cookbook.css" type="text/css" />
<style type="text/css">
table {table-collapse:collapse; border-spacing:0}
td {border:2px groove black; padding:7px; background-color:#ccffcc}
th {border:2px groove black; padding:7px; background-color:#ffffcc}
.ctr {text-align:center}
</style>
<script type="text/javascript">
if (document.images) {
    var imgArray = new Array();
    imgArray[0] = new Image(60,120);
    imgArray[0].src = "digits/0.gif";
    imgArray[1] = new Image(60,120);
    imgArray[1].src = "digits/1.gif"
    imgArray[2] = new Image(60,120);
    imgArray[2].src = "digits/2.gif";
```

продолжение ↗

Листинг 15.3 (продолжение)

```
    imgArray[3] = new Image(60,120);
    imgArray[3].src = "digits/3.gif";
    imgArray[4] = new Image(60,120);
    imgArray[4].src = "digits/4.gif";
    imgArray[5] = new Image(60,120);
    imgArray[5].src = "digits/5.gif";
    imgArray[6] = new Image(60,120);
    imgArray[6].src = "digits/6.gif";
    imgArray[7] = new Image(60,120);
    imgArray[7].src = "digits/7.gif";
    imgArray[8] = new Image(60,120);
    imgArray[8].src = "digits/8.gif";
    imgArray[9] = new Image(60,120);
    imgArray[9].src = "digits/9.gif";
}

var nextYear = new Date().getYear() + 1;
nextYear += (nextYear < 1900) ? 1900 : 0;
var targetDate = new Date(nextYear,0,1);
var targetInMS = targetDate.getTime();
var oneSec = 1000;
var oneMin = 60 * oneSec;
var oneHr = 60 * oneMin;
var oneDay = 24 * oneHr;

function countDown() {
    var nowInMS = new Date().getTime();
    var diff = targetInMS - nowInMS;
    var scratchpad = diff / oneDay;
    var daysLeft = Math.floor(scratchPad);
    // осталось часов
    diff -= (daysLeft * oneDay);
    scratchpad = diff / oneHr;
    var hrsLeft = Math.floor(scratchPad);
    // осталось минут
    diff -= (hrsLeft * oneHr);
    scratchpad = diff / oneMin;
    var minsLeft = Math.floor(scratchPad);
    // осталось секунд
    diff -= (minsLeft * oneMin);
    scratchpad = diff / oneSec;
    var secsLeft = Math.floor(scratchPad);
    // устанавливаем изображения в соответствии со временем
    setImages(daysLeft, hrsLeft, minsLeft, secsLeft);
}

function setImages(days, hrs, mins, secs) {
    var i;
    days = formatNum(days, 3);
    for (i = 0; i < days.length; i++) {
        document.images["days" + i].src = imgArray[parseInt(days.charAt(D)].src;
    }
```

```
    hrs = formatNum(hrs, 2);
    for (i = 0; i < hrs.length; i++) {
        document.images["hours" + i].src = imgArray[parseInt(hrs.charAt(i))].src;
    }
    mins = formatNum(mins, 2);
    for (i = 0; i < mins.length; i++) {
        document.images["minutes" + i].src = imgArray[parseInt(mins.charAt(i))] src;
    }
    secs = formatNum(secs, 2);
    for (i = 0; i < secs.length; i++) {
        document.images["seconds" + i].src = imgArray[parseInt(secs.charAt(i))].src;
    }
1
function formatNum(num, len) {
  | var numStr = "" + num;
   while (numStr.length < len) {
        numStr = "0" + numStr;
    }
    return numStr
}


</script>
</head>
<body style="margin left:10%; margin-right:10%" onload="setInterval('countDown()'  1000)">
<h1>До Нового года осталось</h1>
fchr />

<table cellspacing="5" cellpadding="5">
<tr>
   <td align="right">
   <img name="days0" src="digits/0.gif" height="120" width="60" alt="days">
   <img name="days1" src="digits/0.gif" height="120" width="60" alt="days">
 | <img name="days2" src="digits/0.gif" height="120" width="60" alt="days">
   </td>
   <td align="left">
       <img src="digits/days gif" height="120" width="260" alt="days">
   </td>
 •/tr>
"tr>
 L  <td align="right">
       <img name="hours0" src="digits/0.gif" height="120" width="60"
        alt="hours">
       <img name="hours1" src="digits/0.gif" height="120" width="60"
        alt="hours">
   </td>
   <td align="left">
       <img src="digits/hours.gif" height="120" width="360" alt="hours">
   </td>
 /tr>
"tr>
   <td align="right">
       <img name="minutes0" src="digits/0.gif" height="120" width="60"
        alt="minutes">
```

продолжение ⇙

Листинг 15.3 (продолжение)

```
        <img name="minutes1" src="digits/0.gif" height="120" width="60"
         alt="minutes">
    </td>
    <td align="left">
       <img src="digits/minutes.gif" height="120" width="450" alt="minutes">
    </td>
</tr>
<tr>
    <td align="right">
        <img name="seconds0" src="digits/0.gif" height="120" width="60"
         alt="seconds">
        <img name="seconds1" src="digits/0.gif" height="120" width="60"
         alt="seconds">
    </td>
    <td align="left">
       <img src="digits/seconds.gif" height="120" width="460" alt="seconds">
    </td>
</tr>
</table>

</body>
</html>
```

В показанном примере таймер настроен на начало следующего года в часовом поясе пользователя. На HTML-страницу помещается простейшая таблица с местами для изображений цифр. Так как это приложение может работать в устаревших браузерах, испытывающих трудности с CSS-форматированием, в ячейках таблицы используется старомодный, но все еще поддерживаемый атрибут align, а в элементах **img** — атрибут name.

Сценарий на этой странице начинается с предварительной загрузки изображений, чтобы предотвратить возможные задержки при их первом появлении. При загрузке страницы в кэш загружаются десять изображений различных цифр, все одинаковой высоты и ширины.

Так как некоторые функции этого сценария вызываются многократно, константы из них помещены в глобальные переменные, вычисляемые только один раз. Кроме того, так как код должен работать во всех браузерах, кроме первого поколения, он должен включать обход проблемы 2000 года, с которой сталкивается метод getYear() объекта Date (метод **getFullYear()** появился только в браузерах 4-й версии).

Вычисления с датами производит вызываемая каждую секунду функция countDown(), сравнивающая текущее состояние часов с заданным временем. Она вызывается периодически, потому что для ее запуска используется метод **setInterval()**.

Далее следует функция **setImages()**, устанавливающая значение свойства src для изображений, образующих надпись. Для работы этой функции требуется преобразовать числовые значения, переданные из countDown() в строку. Для этого применяется функция **formatNum()**, которая заодно добавляет к числу ведущие нули, если это необходимо.

Рассматриваемое приложение представляет собой хороший пример ситуации, в которой для управления действием имеет смысл использовать метод setInterval(). В то время как метод setTimeout() предназначен для однократного вызова функции через некоторое время, setInterval() автоматически вызывает функцию с заданной периодичностью. Если предполагается работать со старыми браузерами, не поддерживающими setInterval() (появившийся в браузерах четвертой версии), такое поведение можно имитировать, рекурсивно вызывая countDown() с помощью setTimeout() в последней строке функции:

```
setTimeout("countDown()"  1000):
```

Как и в случае с автоматической прокруткой из рецепта 15.5, периодический вызов функции таймера не всегда работает равномерно. Если несколько секунд понаблюдать за работой этого кода, можно заметить случайную неравномерность при смене цифр. Уменьшив период, скажем, до 100 миллисекунд, можно сделать работу более равномерной, но в результате 10 раз в секунду будет вызываться функция, что может повлиять на работу других сценариев на странице. Можно поэкспериментировать с различными настройками, чтобы найти баланс между плавностью и производительностью.

До сих пор время рассматривалось в локальном часовом поясе пользователя, но могут возникать ситуации, когда необходима привязка к определенной дате и времени в конкретной точке планеты, например к пресс-конференции, посвященной выпуску нового продукта. Чтобы время отсчитывалось правильно, следует производить некоторые дополнительные вычисления, учитывающие часо вые пояса, чтобы пользователи от Австралии до Гренландии видели одинаковое значение таймера.

Для этого нужен несложный код, но хронометраж может оказаться не столь простым. Чтобы реализовать привязку к уникальному моменту времени, следует изменить объявления глобальных переменных с такого:

```
var targetDate = new Date(nextYear.O.l):
var targetInMS = targetDate.GetTime();
```

на такое:

```
var targetlnMS = Date.UTC(nextYear. 0. 1. 0. 0. 0);
```

Такая запись соответствует новогодней полночи по гринвичскому времени (GMT). Если нужны другие дата и время, их значения следует просто вписать в шесть передаваемых методу параметров в таком порядке: год (полный), месяц (отсчет начинается с нуля), день, час, минута и секунда по гринвичскому времени. Например, если вы разместите в сети информацию о веб-трансляции концерта, который пройдет 10 марта 2003 года в 8 часов в Лос-Анджелесе, сначала необходимо определить, какому гринвичскому времени соответствует этот момент. В данном случае Лос-Анджелес находится в Тихоокеанском часовом поясе, и время соответствует Тихоокеанскому поясному времени, которое на 8 часов опережает GMT. Другими словами, когда в Лос-Анджелесе 8 часов вечера, по GMT наступило 4 часа утра следующего дня. Следовательно, нужно настраивать таймер на 4 утра 11 марта (индекс месяца — 2):

```
var targetlnMS = Date.UTC(2003. 2. 11. 4. 0. 0):
```

Существует более простой путь вычисления времени в GMT Установите системное время и часовой пояс в соответствие с тем местом, где должно произойти событие (скорее всего, вы уже находитесь там). Затем запустите следующий несложный калькулятор, чтобы определить дату и время по GMT:

```
<html>
<head>
<title>GMT Calculator</title>
<script type="text/javascript">
function calcGMT(form) {
    var date = new Date(form.year.value, form.month.value, form.date.value,
                        form.hour.value, form.minute.value, form.second.value);
    form.output.value - date.toUTCString();
}
</script>
</head>
<body>
<form>
<b>Enter a local date/time</b><br>
Год:<input type="text" name="year" value="0000"><br>
Месяц (0-11):<input type="text" name="month" value="0"><br>
Дата (1-31):<input type="text" name="date" value="0"><br>
Час (0-23):<input type="text" name="hour" value="0"><br>
Минуты (0-59):<input type="text" name="minute" value="0"><br>
Секунды (0-59):<input type="text" name="second" value="0"><br>
<input type="button" value="Расчет GMT" onclick="calcGMT(this.form)"><br>
GMT: <input type="text" name="output" size="60">
</body>
</html>
```

Если для применения этого калькулятора вы изменили настройки системного времени на компьютере, не забудьте затем вернуть их обратно на ваше локальное время.

Иногда необходим другой тип таймера для коротких интервалов времени, применяемый в таких приложениях, как опрос учеников, где нужно сделать что-нибудь или перейти на другую страницу прежде, чем истечет заданный промежуток времени. В таких ситуациях обычно принято показывать на странице, скажем, 60-секундный таймер.

Приведенный ниже код является модификацией сценария из листинга 15.3. Он помещает на странице отсчитывающий время таймер, время которому (в секундах) задает параметр, передаваемый основной функции:

```
// глобальные переменные
var targetInMS, timerInterval;
var oneSec = 1000;

// передача числа секунд для обратного отсчета
function startTimer(secs) {
    var targetTime = new Date();
    targetTime.setSeconds(targetTime.getSeconds() + secs);
    targetInMS - targetTime.getTime();
    // показываем начальные изображения
```

```
    setImages(secs);
    timerInterval = setInterval("countDown()", 100);
}

function countDown() {
    var nowInMS = (new Date()).getTime();
    var diff = targetInMS - nowInMS;
    if (diff <= 0) {
        clearInterval(timerInterval);
        alert("Время истекло!");
        // дополнительная обработка
    } else {
        var scratchpad = diff / oneSec;
        var secsLeft = Math.floor(scratchPad);
        setImages(secsLeft);
    }
}
function setImages(secs) {
    var i;
    secs = formatNum(secs, 2);
    for (i = 0; i < secs.length; i++) {
        document.images["seconds" + i].src =
            imgArray[parseInt(secs.charAt(i))].src;
    }
}
function formatNum(num, len) {
    var numStr = "" + num;
    while (numStr.length < len) {
        numStr = "0" + numStr;
    }
    return numStr
}
```

При необходимости данный код легко можно расширить, чтобы показывать не только секунды, но и минуты.

Смотрите также

Как показать на странице число дней до рождества, рассказано в рецепте 15.7. Рецепты 2.9 и 2.10 посвящены объекту Date и его методам.

15.9. Как указать дату с помощью календаря

NN6 IE5(Win)

Задача

Чтобы помочь пользователям ввести в форму нужную дату, необходимо реализовать всплывающий календарь.

Решение

Интерфейс календаря можно выполнить с помощью таблицы, помещенной в абсолютно позиционируемый div. Сценарий, скрытый за этим элементом управления, должен решать две основные задачи:

- заполнять календарь датами выбранного месяца в выбранном году;
- заполнять поле даты в основной форме по щелчку на календаре.

В обсуждении приведен пример реализации такого элемента управления, включая таблицы стилей и сценарии. Для вызова календаря служит одна из функций из листинга 15.4, называемая showCalendar(). Она должна вызываться одним из элементов управления в форме.

Все необходимые для инициализации процедуры, включая инициализацию библиотеки DHTML API из рецепта 13.3, вызывает обработчик события onload элемента body:

```
<body onload="fillYears(); populateTable(document.dateChooser);
    initDHTMLAPI()">
```

Обсуждение

Это решение содержит большое количество кода как HTML и CSS, так и сценариев. Но и количество решаемых кодом задач соответственно велико: динамически формируется содержимое календаря, устанавливается положение календаря на экране и его видимость, данные из календаря передаются обратно в основную страницу, и решаются многие другие задачи. Пример использования календаря на простой форме для ввода даты показан на рис. 15.3.

Начнем с двух сегментов HTML-кода. Первый из них задает форму для ввода даты, получающую значение из календаря. Щелчок на кнопке (элементу input) вызывает функцию, показывающую календарь:

```
<form name="mainForm" id="mainForm" method="POST" action="">
<p>Введите дату:</p>
<p>мм:<input type="text" name="month" id="month" size="3" maxlength="2" value="1">
flfl.<input type="text" name="date" id="date" size="3" maxlength="2" value="1">
rrrr:<input type="text" name="year" id="year" size="5" maxlength="4" value="2003">
<input type="button" id="showit" value="Календарь »" onclick="showCalendar(event)">
</p>
</form>
```

Другая важная часть HTML описывает позиционируемый контейнер div, содержащий таблицу календаря. Эта таблица доставляется с сервера незаполненной (и скрытой от пользователя) и заполняется сценарием при инициализации. В коде страницы присутствуют только заголовки, обозначающие дни недели и элемент select для выбора месяцев. Названия месяцев из списка в select используются позднее, чтобы поместить название в календарь:

```
<div id="calendar">
<table id="calendarTable" border=1>
```

```
<tr>
    <th id="tableHeader" colspan="7"></th>
</tr>
<tr><th>Sun</th><th>Mon</th><th>Tue</th><th>Wed</th>
<th>Thu</th><th>Fri</th><th>Sat</th></tr>
<tbody id="tableBody"></tbody>
<tr>
    <td colspan="7">
    <p>
    <form name="dateChooser">
        <select name="chooseMonth"
        onchange="populateTable(this.form)">
            <option selected>Январь<option>Февраль
            <option>Март<option>Апрель<option>Май
            <option>Июнь<option>Июль<option>Август
            <option>Сентябрь<option>Октябрь
            <option>Ноябрь<option>Декабрь
    </select>
    <select name="chooseYear" onchange="populateTable(this form)"
    </select>
    </form>
    </p>
    </td>
</tr>
</table>
</div>
```

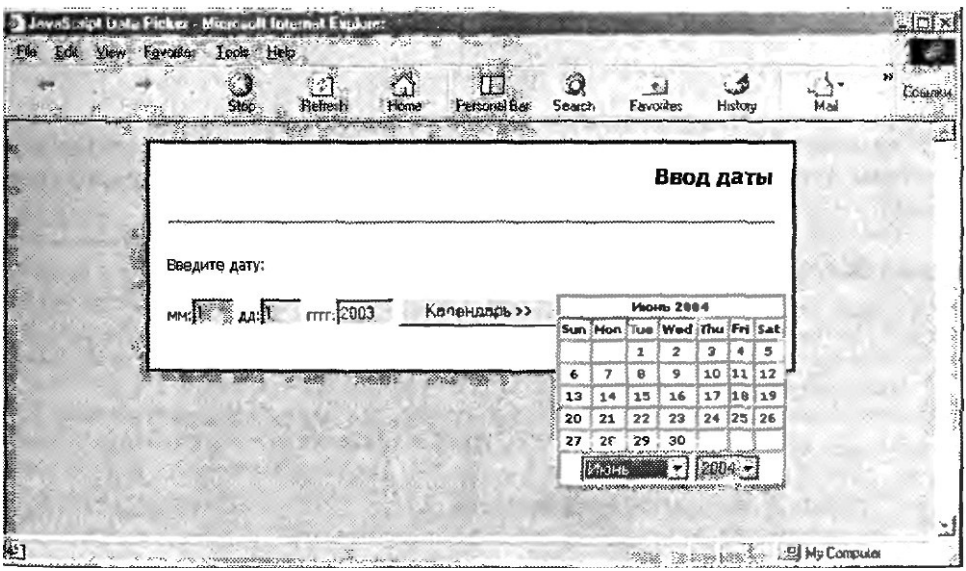


Рис. 15.3. Всплывающий календарь для ввода дать!

Оформление таблицы и ее составных частей определяется довольно сложными таблицами стилей, задающими позиционирование, видимость, выравнивание текста в ячейках, цвета фона, шрифты и т. п. Эффект изменения цвета при

наведении мыши на числа в календаре создается исключительно с помощью псевдоклассов CSS:

```
<style type="text/css">
#calendar {position:absolute;
           left:0px;
           top:0px;
           visibility:hidden
          }
table {font-family:Verdana, Arial, Helvetica  sans-serif;
       background-color:#999999
      }
th {background-color:#ccffcc;
    text-align:center;
    font-size:10px;
    width:26px
   }
#tableHeader {background-color:#ffcccc;
              width:100%
             }
td {background-color:#ffffcc;
    text-align:center;
    font-size:10px
   }
#tableBody tr td {width:26px}
#today {background-color:#ffcc33}
a:link {color:#000000; text-decoration:none}
a.active {color:#000000; text-decoration:none}
a:visited {color:#0000 0; text-decoration:none}
a:hover {color:#990033; text-decoration:underline}
</style>
```

В листинге 15.4 приведены сценарии этого рецепта, а также код для подключения библиотеки DHTML API. Сценарий календаря разделен на четыре секции.

Листинг 15.4. Сценарии для всплывающего календаря

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

      Вспомогательные функции

// день недели первого дня месяца
function getFirstDay(theYear, theMonth){
    var firstDate = new Date(theYear,theMonth,1);
    return firstDate.getDay();
}
// число дней в месяце
function getMonthLen(theYear, theMonth) {
    var nextMonth = new Date(theYear, theMonth + 1, 1);
    nextMonth.setHours(nextMonth.getHours() - 3);
    return nextMonth.getDateO;
}
```

```
function getElementPosition(elemID) {
    var offsetTrail = document.getElementById(elemID);
    var offsetLeft = 0;
    var offsetTop = 0;
    while (offsetTrail) {
        offsetLeft += offsetTrail.offsetLeft;
        offsetTop += offsetTrail.offsetTop;
        offsetTrail = offsetTrail.offsetParent;

    if (navigator.userAgent.indexOf("Mac") != -1 &&
        typeof document.body.leftMargin != "undefined") {
        offsetLeft += document.body.leftMargin;
        offsetTop += document.body.topMargin;
    }
    return {left:offsetLeft, top:offsetTop};
}
// позиционирование и отображение календаря
function showCalendar(evt) {
    evt = (evt) ? evt : event;
    if (evt) {
      if (document.getElementById("calendar").style.visibility!="visible"){
            var elem = (evt.target) ? evt.target : evt.srcElement;
            var position = getElementPosition(elem.id);
            shiftTo("calendar", position.left + elem.offsetWidth,
                position.top);
            show("calendar");
      } else {
            hide("calendar");
      }
    }
}
```

Рисование содержимого календаря

```
// Очистка и повторное заполнение таблицы на основе выбора в форме
function populateTable(form) {
    // определяем дату, введенную в форме
    var theMonth = form.chooseMonth.selectedIndex;
    var theYear = parseInt(form.chooseYear.options[form.chooseYear.selectedIndex].text);
    // инициализация зависящих от даты переменных
    var firstDay = getFirstDay(theYear, theMonth);
    var howMany = getMonthLen(theYear, theMonth);
    var today = new Date();

    // заполнение заголовка таблицы, содержащего название месяца и год
    document.getElementById("tableHeader").innerHTML =
        form.chooseMonth.options[theMonth].text + " " + theYear;

    // инициализация переменных для создания таблицы
    var dayCounter = 1;
    var TBody = document.getElementById("tableBody");
```

продолжение

Листинг 15.4 (продолжение)

```
    // очистка существующих строк таблицы
    while (TBody.rows.length > 0) {
        TBody.deleteRow(O):
    }
    var newR. newC. dateNum:
    var done=false:
    while (!done) {
        // создаем новую строку в конце таблицы
        newR = TBody.insertRow(TBody.rows.length);
        if (newR) {
            for (var i = 0: i < 7: i++) {
                // создаем новую ячейку в конце строки
                newC = newR.insertCell(newR.cells.length):
                if (TBody.rows.length == 1 && i < firstDay) {
                    //до первого дня клетки должны быть пусты
                    newC.innerHTML = " :"
                    continue:
                }
                if (dayCounter == howMany) {
                    // после этой строки новых быть не должно
                    done = true:
                }
                // помещаем в ячейку число и ссылку (для ячеек после
                // последнего дня - ничего)
                if (dayCounter <= howMany) {
                    if (today.getFullYear() == theYear &&
                        today.getMonth() == form.chooseMonth.selectedIndex
                            && today.getDate() == dayCounter) {
                      • newC.id - "today":
                    }
                    newC.innerHTML = "<a href='#'onclick='chooseDate(" +
                        dayCounter + "." + theMonth + "." + theYear +
                        " ) : return false;'>" + dayCounter + "</a>":
                     dayCounter++;
                } else {
                    newC.innerHTML = " ":
                }
            }
        } else {
            done - true:
        }
    }
}
/*******************
   Инициализация
*******************/
// формирование динамического списка для выбора года
function fill Years O {
    var today = new Date():
    var thisYear - today.getFullYear():
    var yearChooser = document.dateChooser.chooseYear:
    for (i = thisYear: i < thisYear + 5: 1++) {
        yearChooser.options[yearChooser.options.length] = new Option(i  i):
```

```
    }
    setCurrMonth(today);

// делаем текущий месяц выбранным
function setCurrMonth(today) {
    document.dateChooser.chooseMonth.selectedIndex = today.getMonth();
}

  Обработка выбора

function chooseDate(date, month, year) {
    document.mainForm.date.value = date;
    document.mainForm.month.value = month + 1;
    document.mainForm.year.value = year;
    hide("calendar");
}
</script>
```

Первая секция сценария содержит несколько вспомогательных функций, необходимых для работы остальной части кода. Первая пара функции, getFirstDay() и **getMonthLen()**, применяется в процедурах формирования календаря для того, чтобы определить день недели первого дня заданного месяца, а затем — число дней в месяце. Трехчасовая коррекция в **getMonthLen()** нужна для учета аномалий вычислений даты, случающихся, когда месяц приходится на переход с лета на зиму. Цель в том, чтобы получить номер дня, предшествующего первому дню следующего месяца.

Когда наступает пора показать календарь, в игру вступает следующая пара функций. Первая из них, getElementPosition() (см. рецепт 13.8), определяет положение элемента (в данном случае — кнопки Календарь >>). Эти координаты затем использует следующая функция, showCalendar(), для размещения календаря справа от кнопки перед тем, как его показать. Для управления положением и видимостью элементов используются функции DHTML API.

Вычисляет даты и формирует HTML-код, образующий календарь таблицы, центральная функция этого приложения, **populateTable()**. Ее работа начинается со сбора основных сведений из списков select в нижней части календаря. Затем она помещает название месяца и год в заголовок таблицы. После некоторых связанных с DOM вспомогательных действий сценарий очищает таблицу от предыдущего содержимого. В центре сценария лежит цикл while, добавляющий в таблицу новые строки. Для каждой новой строки цикл for формирует ячейки, образующие семь столбцов, и заполняет эти ячейки датами и ссылками, служащими для передачи данных обратно в главную форму. Один небольшой штрих: идентификатор ячейки, соответствующей текущему дню, назначается так, чтобы к ней был применен отдельный стиль, выделяющий эту ячейку среди остальных.

Две процедуры инициализации, fillYears() и **setCurrMonth()**, подготавливают элементы select в нижней части календаря, чтобы список лет обновлялся по мере хода времени. Кроме того, при первом появлении календаря списки настраиваются на текущий год и месяц.

Когда пользователь щелкает мышью на одной из дат, обработчик события вызывает функцию chooseDate(), в которую в качестве параметров передаются число, месяц и год. Значения этих параметров записываются в обработчик в процессе формирования HTML-кода в populateTable(). Получив значения, функция chooseDate() распределяет их по трем полям ввода в исходной форме.

Всплывающий календарь работает в Internet Explorer 5 для Windows или старше, а также в Netscape 6 или старше. К сожалению, ошибки в реализации манипуляций с таблицами в IE 5 для Mac делают невозможной работу приложения в этом браузере.

Netscape (по крайней мере, вплоть до 7-й версии) демонстрирует необычную медлительность, когда div с календарем перекрывает элементы формы, особенно ее текстовые поля.

Конфликты в процессе отображения приводят к тому, что текстовое поле вытесняет позиционируемый элемент, так что становится невозможным щелкнуть на дате в календаре или активизировать элемент select, если они лежат поверх текстового поля. Поэтому следует постараться так расположить все элементы страницы, чтобы div не контактировал с элементами управления формы. В IE подобных проблем не возникает.

Внешний вид видимой части приложения по большей части определяется таблицами стилей. Взяв за основу идентификаторы и HTML-теги базового календаря, можно в широких пределах менять его оформление. Если следовать принятым соглашениям об именах, код для генерации и модификации календаря будет работать без проблем.

Вам может показаться привлекательной идея реализовать возможность перетаскивать календарь за полосу заголовка. Чтобы добавить к рецепту подобную функциональность, можно адаптировать код перетаскивания элементов из рецепта 13.11.

Смотрите также

Подробности об использовании объектов даты ищите в рецептах 2.9 и 2.10. Как скрывать и отображать элементы, показано в рецепте 12.7. Библиотека DHTML API рассматривается в рецепте 13.3. Рецепт 13.8 рассказывает о том, как получить координаты элементов документа.

15.9. Анимированный индикатор выполнения

NN6 **IE5**

Задача

Когда начинается выполнение длительной операции, необходимо отобразить на странице индикатор процесса выполнения.

Решение

Предлагаемое решение содержит минимальный HTML-код, таблицу стилей для придания коду внешнего вида и сценарий, управляющий анимацией. Имеющийся на странице HTML-код состоит из группы вложенных друг в друга div, каждый из которых образует отдельную часть интерфейса пользователя: рамку вокруг индикатора выполнения, текст подписи, пустой индикатор и закрашенный индикатор, в процессе анимации растущий вправо:

```
<div id="progressBar">
    <div id="progressBarMsg">Вычисление. .</div>
    <div id="sliderWrapper">0%
        <div id="slider">0%</div>
    </div>
</div>
```

При загрузке страницы индикатор не отображается, а в остальном его вид контролируется сложными таблицами стилей, описанными в обсуждении. Изменяя эти стили, можно настраивать внешний вид под свои нужды.

Процесс управления индикатором состоит из вызовов функций showProgressBar(), calcProgressBar() и hideProgressBar(), приведенных в листинге 15.5. Как с помощью многократных вызовов calcProgressBar() создать анимированный индикатор, рассказывается в обсуждении. Для инициализации страницы обработчик события onload должен вызывать две процедуры инициализации: DHTML API (см. рецепт 13.3) и функцию initProgressBar():

```
<body onload="initDHTMLAPI(); initProgressBar()">
```

Обсуждение

Несмотря на простоту HTML-кода индикатора, он может содержать значительное количество информации. На рис. 15.4 показано, как выглядит индикатор в середине процесса.

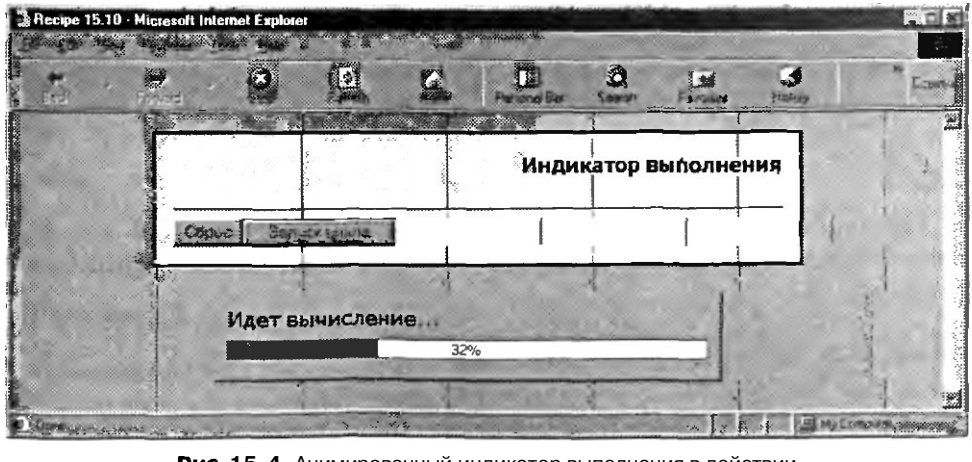


Рис. 15.4. Анимированный индикатор выполнения в действии

Вид полосы индикатора контролирует следующая таблица стилей:

```
<style type="text/css">
#progressBar {position:absolute;
              width:400px;
              height:35px;
              visibility:hidden;
              background-color:#99ccff;
              padding:20px;
              border-width:2px;
              border-left-color:#9999ff;
              border-top-color:#9999ff;
              border-right-color #666666;
              border-bottom-color:#666666;
              border-style:solid;
             }
#progressBarMsg {position:absolute;
                 left:10px;
                 top:10px;
                 font:18px Verdana. Helvetica  sans-serif bold
                }
#sliderWrapper {position:absolute;
                left:10px;
                top:40px;
                width:417px;
                height:15px;
                background-color:#ffffff;
                border:1px solid #000000;
                text-align:center;
                font-size:12px
               }
#slider{position:absolute;
        left:0px;
        top:0px;
        width:420px;
        height:15px;
        clip:rect(0px 0px 15px 0px);
        background-color:#666699;
        text-align:center;
        color:#ffffff;
        font-size:12px
       }
</style>
```

Управляющий индикатором сценарий, показанный в листинге 15.5, основан на использовании библиотеки DHTML API из рецепта 13.3 и функции centerWindow() из рецепта 13.7.

Листинг 15.5. Управляющий индикатором сценарий

```
<script type="text/javascript" src="../js/DHTMLAPI.js"></script>
<script type="text/javascript">
// Центрирование позиционируемого элемента, идентификатор которого передан
// в качестве параметра, по отношению к окну или фрейму, и его отображение
```

```
function centerOnWindow(elemID) {
    // 'obj' - позиционируемый объект
    var obj - getRawObject(elemID);
    // положения прокрутки окна
    var scrollX = 0. scrollY = 0:
    if (document. body && typeof document. body. scroll Top != "undefined") {
        scrollX += document.body.scroll Left:
        scrollY += document.body.scroll Top:
        if (document.body.parentNode &&
            typeof document.body.parentNode.scrollTop != "undefined") {
            scrollX += document.body.parentNode.scrollLeft;
            scroll Y += document.body.parentNode.scrollTop
        }
    } else if (typeof window.pageXOffset != "undefined") {
        scrollX += window.pageXOffset;
        scrollY += window.pageYOffset;
    }
    var x - Math.round((getInsideWindowWidth()/2) - (getObjectWidth(obj)/2))
            + scrollX;
    var y - Math.round((getInsideWindowHeight()/2) -
            (getObjectHeight(obj)/2)) + scrollY;
    shiftTo(obj. x. y) :
    show(obj):
}

function initProgressBarO {
    // Создание объекта, свойства которого по умолчанию (в CSS-совместимом
    // режиме) равны нулю. Действительные значения для обычного режима будут
    // записаны в него позднее.
    if (navigator.appName == "Microsoft Internet Explorer" &&
        navigator.userAgent.indexOt("Win") != -1 &&
        (typeof document.compatMode == "undefined" ||
        document.compatMode == "BackCompat")) {
        document.getElementById("progressBar").style.height - "81px":
        document.getElementById("progressBar").style.width = "444px";
        document.getElementById("sliderWrapper").style.fontSize =
            "xx-small";
        document.getElementById("slider").style.fontSize = "xx-small";
        document.getElementById("slider").style.height - "13px":
        document.getElementById("slider").style.width = "415px";
    }
}

function showProgressBar() {
    centerOnWindow("progressBar");
}

function calcProgress(current. total) {
    if (current <= total) {
        var factor = current/total:
        var pct - Math.ceil(factor * 100):
        document.getElementById("sliderWrapper").firstChild nodeValue - pct
            + "%".
```

продолжение

Листинг 15.5 (продолжение)

```
        document getElementById("slider").firstChild.nodeValue - pct + "%";
        document.getElementById("slider").style.clip - "rect(0px " +
            parseInt(factor * 417) + "px 16px 0px)";
    }
}

function hideProgressBar() {
    hide( progressBar"):
    calcProgress(0, 0):
}
</script>
```

Функция centerOnWindow() вызывается каждый раз, когда показывается индикатор выполнения (в функции showProgressBar()). Таким образом, даже если пользователь изменит размер окна браузера в процессе анимации, индикатор все равно останется по центру.

По причине широкого использования рамок и отступов для определения внешнего вида индикатора, настройка размера вложенного div требует небольшого дополнительного кодирования. Несовместимость между разными режимами совместимости IE сильно влияет на определение различных размеров. Описание, имеющееся в таблице стилей, рассчитано на режим CSS-совместимости и работает «как есть» в Netscape 6 и старше, а также в IE 5 для Macintosh. Для IE версий 5, 5.5 и 6, работающего в режиме обратной совместимости, функция initProgressBar() подстраивает некоторые основные размеры.

Видимость и анимацию индикатора контролируют три функции. Две из них просто показывают или скрывают индикатор, а третья, calcProgress(), является ключевой для анимации. Индикатор изменяет свой вид благодаря изменению размера отсекающего прямоугольника у наиболее глубоко вложенного div. Изначально он задается с нулевой шириной. Все вычисления основываются на сравнении текущего значения некоторой величины с ее максимальным значением, это может быть применено к любой делимой операции. Полученное отношение определяет положение левой границы прямоугольника отсечения.

Чтобы поэкспериментировать с индикатором, были созданы несколько проверочных функций, просто повторяющих некоторую последовательность вычислений с небольшой нерегулярностью, чтобы сделать работу интереснее. Глобальный объект по имени loopObject имеет несколько вспомогательных свойств, помогающих вычислениям. Примером функции, выигрывающей от использования подобного индикатора, служит функция runProgressbar(). Ключ к успешному применению рецепта — повторяющийся сценарий, запускаемый через setTimeout() или setTimeout(), чтобы страница могла обновляться между итерациями. Когда вычисления будут завершены, следует повторно вызвать calcProgress() с теми же параметрами, чтобы на индикаторе ненадолго появилось значение 100 % перед тем, как он исчезнет.

Определить тип операций, к которым можно применить индикатор выполнения, может быть не так просто. Важно, чтобы операция заняла значительное время и индикатор успел отобразиться. Но не менее важным является требование, чтобы операция происходила уже после того, как страница загружена, образую-

щие индикатор элементы div были уже загружены, а их размеры были подстроены в соответствии с режимом работы IE для Windows.

Кроме того, для операции вы должны знать некоторое конечное значение, чтобы сравнивать с ним текущий прогресс. Одна из возможных задач — это индикация процесса предварительной загрузки изображений, запускаемого обработчиком события onload. Ваш эквивалент функции runProgressBar() должен перебирать массив объектов image, проверяя у каждого объекта, равно ли true значение свойства complete. Если это так, индикатор продвигается на величину, соответствующую одному изображению, а функция должна перейти к следующему элементу массива с помощью setTimeout(). Если значение равно false, следует повторно проверить то же изображение, передав через setTimeout() прежнее значение индекса. Конечно, изображения не всегда прибывают с сервера в том порядке, в котором они объявлены в коде, поэтому индикатор может не соответствовать процессу. Кроме того, если изображения уже находятся в кэше, цикл по всем элементам массива может только задержать пользователя при обращении к конечной странице.

Индикатор выполнения — хорошее решение для тех ситуаций, когда пользователь может потерять терпение, в то время как выполняются скрытые операции. Если во время этого промежутка развлекать пользователя, он становится гораздо терпеливее.

Смотрите также

В рецепте 12.7 рассказывается, как скрывать и отображать элементы. Как сделать элемент позиционируемым, описано в рецепте 13.1. Подробности о библиотеке DHTML API смотрите в рецепте 13.3. Центрирование позиционируемого элемента относительно окна или фрейма рассматривается в рецепте 13.7.

A Коды клавиш клавиатурных событий

В современных браузерах клавиатурные сообщения предоставляют информацию о клавишах и в тех случаях, когда это возможно, о соответствующем символе. Значения символов могут быть определены в событии onkeypress. В таблице приведены коды для символов нижней части таблицы ASCII. Некоторые из этих кодов относятся к специальным клавишам (клавиша возврата Backspace и табуляции Tab).

Символ	Значение	Символ	Значение
Backspace (возврат)	8	.	46
Tab	9	/	47
Enter (Return на Mac)	13	0	48
Space (пробел)	32	1	49
!	33	2	50
"	34	3	51
#	35	4	52
$	36	5	53
%	37	6	54
&	38	7	55
'	39	8	56
(	40	9	57
)	41	:	58
*	42	;	59
+	43	<	60
,	44	=	61
-	45	>	62

Символ	Значение
?	63
@	64
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	7
I	73
J	74
K	75
L	76
ntblM	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
[	91
\	92
	93
	94
	95

Символ	Значение
.	96
a	7
b	98
c	99
d	100
e	101
f	102
g	103
h	104
i	105
i	106
k	107
l	108
m	109
п	110
0	111
P	112
q	1
r	4
S	1
t	116
u	117
V	118
W	119
x	1
y	121
z	122
{	123
|	4
}	125
	126
Delete	127

Б Коды клавиш

Коды клавиш — это числовые значения, обозначающие определенные клавиши и не обязательно соответствующие какому-либо символу. Например, клавише A на клавиатуре всегда соответствует один и тот же код, хотя код символа может быть различным, в зависимости от того, нажата ли клавиша Shift: — 65 для A или 97 для a. На коды клавиш не влияет состояние клавиш-модификаторов. Код символа (см. приложение А) можно определить из обработчика события onkeypress, в то время как коды клавиш, включая управляющие и функциональные клавиши, доступны для обработчиков событий onkeydown и onkeyup. В таблице перечислены все клавиши типичной клавиатуры и их коды.

Клавиша	Код	Клавиша	Код
Alt	18	F5	116
Курсор вниз	40	F6	117
Курсор влево	7	F7	118
Курсор вправо	39	F8	119
Курсор вверх	38	F9	120
Backspace (возврат)	8	F10	121
Caps Lock	20	F11	122
dctlparCtrl	17	F12	123
Delete	4	Home	36
End	35	Insert	45
Enter	13	Num Lock	144
Esc	27	(цифровые клавиши) -	109
F1	112	(цифровые клавиши) *	106
F2	113	(цифровые клавиши) .	110
F3	1	(цифровые клавиши) /	111
F4	115	(цифровые клавиши) +	107
(цифровые клавиши) 0	96	P	80

Клавиша	Код	Клавиша	Код
(цифровые клавиши) 1	97	Q	81
(цифровые клавиши) 2	98	R	82
(цифровые клавиши) 3	99	S	83
(цифровые клавиши) 4	100	T	84
(цифровые клавиши) 5	101	U	85
(цифровые клавиши) 6	102	V	86
(цифровые клавиши) 7	103	W	87
(цифровые клавиши) 8	104	X	88
(цифровые клавиши) 9	105	Y	89
Page Down	34	Z	90
PageUp	33	1	49
Pause	19	2	50
Print Scrn	44	3	51
Scroll Lock	145	4	52
Shift	16	5	53
Пробел	32	6	54
Tab	9	7	55
A	65	8	56
B	66	9	57
C	67	0	48
D	68	,	222
E	69	-	189
F	70		188
G	71		190
H	72	/	191
I	73	`	186
J	74	[	219
K	75	\	220
L	76	]	221
M	77	/	192
N	78	=	187
0	79		

B Зарезервированные слова ECMAScript

Все приведенные в таблице слова зарезервированы интерпретатором ECMAScript, встроенным в поддерживающие сценарии браузеры. Не следует использовать эти слова в качестве идентификаторов переменных, функции или объектов. Большая часть этих слов уже занята существующими интерпретаторами JavaScript, а остальные могут появиться в будущих версиях.

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

[фавитный указатель

И

К

М

H

O

Дэнни Гудман

JavaScript и DHTML. Сборник рецептов. Для профессионалов

Перевел с английского Д. Шинтяков

Главный редактор	*Е. Строганова*
Заведующий редакцией	*И. Корнеев*
Руководитель проекта	*В. Шрага*
Литературный редактор	*С. Карпенок*
Иллюстрации	*В. Шендерова*
Художник	*Н. Биржаков*
Корректор	*В. Листова*
Верстка	*Р. Гришанов*

Лицензия ИД № 05784 от 07.09.01.

Подписано к печати 17.03.04 Формат 70x100/16. Усл. п. л. 42,57.
Тираж 3500. Заказ 608
ООО «Питер Принт», 196105, Санкт-Петербург, ул. Благодатная, д. 67в.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2;
95 3005 — литература учебная.

Отпечатано с готовых диапозитивов
в ФГУП ордена Трудового Красного Знамени «Техническая книга»
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций
190005, Санкт-Петербург, Измайловский пр., 29.

O REILLY®

JavaScript и DHTML Сборник рецептов

ДЛЯ ПРОФЕССИОНАЛОВ

‹Как сде гать чтобы? > Большинство вопросов задаваемых на многочисленных форумах в Интернете начинаются именно с этих слов Книга которую вы держите Б руках отвечает как раз на такие вопросы За долгие годы работы автор прочитал десятки тысяч дискуссии на форумах посвященных языкам программирования и составил список самых популярных вопросов задаваемых программистами разного уровня Создавая свою книгу готовых решении Дэнни Гудман опирался на самые современные стандарты ECMA и W3C DOM

Это книга о < вкусных и здоровых сценариях которые сделают веб-страницу индивидуальной и при этом качественной Здесь собраны самые насущные вопросы и практические примеры с которыми неизбежно сталкиваются все те кто всерьез занимается написанием саитов с использованием JavaScript и DHTML Перед вами настоящая «поваренная книга веб-программиста. на все поставленные в книге вопросы даются четкие ответы в виде сценариев с подробными пояснениями. Эти сценарии без труда можно вставить в собственные приложения их код вы наидете не только в книге но и на саите издательства по адресу www.piter.com.

В этом сборнике собраны рецепты разного уровня от самых простых помогающих работать со строками и вычислять даты на JavaScript до полноценных библиотек решающих такие сложные задачи как кросс браузерное позиционирование HTML-элементов и сортировка таблиц. Всего в книге содержится более 150 рецептов охватывающих следующие темы

- работа с интерактивными формами и таблицамистилеи·
- создание удобной навигации повеб-странице·
- разработка динамического контента
- создание визуальных эффектов для статического контента,
- позиционирование HTML-элементов-
- управление окнами браузера ифреимами

Дэнни Гудман пишет о персональных компьютерах и бытовои электронной аппаратуре с конца 70-х годов Его перу принадлежат около трех десятков книг, среди которых такие бестселлеры как *Dynamic HTML: The Definitu e Reference* и *The JaaaScript Bible* Кром оі о он является консультантом по созданию клиентских сценариев для веб-саитов

Посетите web-сайт издательства O'Reilly: www.oreilly.com

ПИТЕР®

Серия: *для профессионалов*

Уровень пользователя: *опытный*

ISBN 5-94723-817-9