# PHP и jQuery

## ДЛЯ ПРОФЕССИОНАЛОВ

# Pro PHP and jQuery

Jason Lengstorf

Apress®

# PHP и jQuery

## ДЛЯ ПРОФЕССИОНАЛОВ

Джейсон Ленгсторф


Москва • Санкт-Петербург • Киев
2011

ББК 32.973.26-018.2.75
Л44
УДК 681.3.07

Издательский дом "Вильямс"

Главный редактор *С.Н. Тригуб*

Зав. редакцией *В.Р. Гинзбург*

Перевод с английского и редакция канд. хим. наук А.Г. Гузикевича

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

**Ленгсторф**, Джейсон.
Л44 PHP и jQuery для профессионалов. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 352 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1693-8 (рус.)

**ББК 32.973.26-018.2.75**



Научно-популярное издание

**Джейсон Ленгсторф**

**PHP и jQuery для профессионалов**

Литературный редактор *Е.Д. Давидян*
Верстка *М.А. Удалов*
Художественный редактор *В.Г. Павлютин*
Корректор *Л.А. Гордиенко*

Подписано в печать 25.10.2010. Формат 70x100/16. Гарнитура Times. Печать офсетная
Усл. печ. л. 28,38. Уч.-изд. л. 15,1 Тираж 1500 экз. Заказ № 0000

Отпечатано по технологии CtP
в ОАО "Печатный двор" им. А. М. Горького. 197110, Санкт-Петербург, Чкаловский пр., 15

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1693-8 (рус.)
ISBN 978-1-4302-2847-9 (англ.)

# Оглавление

# Содержание

# Об авторе

**Джейсон Ленгсторф** — веб-дизайнер и разработчик, проживающий в штате Монтана. Его специализация — создание программного обеспечения для управления веб-контентом на основе PHP, MySQL, AJAX и веб-стандартов. Большую часть времени отдает своей компании Ennui Design, интернациональный коллектив которой разрабатывает первоклассные веб-сайты. Он также управляет собственной фирмой по пошиву одежды Humblecock, пытаясь при этом выкроить время для своих хобби, таких как гольф и путешествия.

# О рецензенте

**Роберт Бан** (Robert Banh) — опытный разработчик, занимающийся программированием чуть ли не с самого детства. Специализируется на разработке веб-приложений с использованием PHP/MySQL, Zend и CodeIgniter для крупных заказчиков, таких как IBM, HP, Unisys и KLRU, а также для общественных организаций. В настоящее время работает в Техасском университете в Остине, где получил ученую степень в области информационных технологий.

# Часть I

# Основные сведения о jQuery

В этой части книги вы познакомитесь с историей появления технологии jQuery и ее базовыми возможностями. Вы получите общее представление о концепциях jQuery, а после изучения материала по объектно-ориентированным средствам PHP, изложенного в части II, сможете выполнить упражнения, приведенные в части III (где разрабатывается проект реального приложения с использованием jQuery и PHP).

Глава 1

# Введение в технологию jQuery

Чтобы объяснить роль технологии jQuery и ее приложений в современном веб-программировании, целесообразно немного остановиться на том, как она возникла, зачем создавалась и что представляло собой программирование на JavaScript до появления jQuery.

В этой главе вы узнаете о том, что такое библиотеки JavaScript, какие задачи они должны решать и почему библиотека jQuery завоевала признание большинства веб-разработчиков. Вы также изучите основы jQuery, разобравшись с тем, как сделать эту библиотеку доступной для своих приложений и как работает ее ядро — мощный селекторный движок.

## К вопросу о выборе jQuery

JavaScript считается довольно неудобным языком для написания веб-приложений. Отсутствие унифицированной поддержки различными браузерами, трудности отладки и пугающий синтаксис способны лишить новичков веры в возможность успешного освоения этого языка.

Справедливости ради следует отметить, что элементы, создающие видимые трудности, в то же время придают мощь этому языку, хотя вряд ли данный аргумент сделает JavaScript более привлекательным для начинающих веб-разработчиков и побудит их добавить этот язык в свой рабочий арсенал.

### Библиотеки JavaScript

Процесс изучения JavaScript всегда был болезненным для разработчиков, и по мере того, как всеобщее чувство недовольства нарастало, некоторые из них взялись создавать **библиотеки** JavaScript, которые часто называют **JavaScript-фреймворками**.

Библиотеки этого типа призваны упростить использование языка JavaScript и сделать его более доступным как для новичков, так и для разработчиков со стажем за счет предоставления простых в применении функций, облегчающих решение повседневных задач. Эффективность библиотек становится особенно заметной при работе с *асинхронным JavaScript и XML* (AJAX), поскольку решение соответствующих задач с применением непосредственно JavaScript оказывается более сложным.

Библиотеки JavaScript предоставляют упрощенный синтаксис для решения типовых задач, что позволяет сократить сроки разработки и ускорить процесс обучения новичков. Кроме того, они частично упрощают написание сценариев JavaScript, не зависящих от типа браузера, беря на себя всю работу по проверке совместимости

с помощью собственных встроенных средств, что обеспечивает *огромную* экономию времени на этапе написания программного кода.

**Примечание.** Подробнее о различиях в использовании AJAX и непосредственно JavaScript говорится в главе 2.

Существует довольно много библиотек JavaScript. Самыми популярными из тех, которые используются разработчиками в настоящее время, являются библиотеки Prototype (`http://www.prototypejs.org`), MooTools (`http://mootools.net`), Yahoo! UI Library (`http://developer.yahoo.com/ui`), а также главный предмет рассмотрения данной книги — jQuery.

## Преимущества, обеспечиваемые jQuery

Каждой библиотеке JavaScript свойственны свои преимущества. Не составляет исключения и библиотека jQuery, которая обладает следующими достоинствами:

- небольшой размер файла (около 23 Кбайт для версии 1.4);
- чрезвычайно простой синтаксис;
- возможность объединения последовательно вызываемых методов в цепочки;
- простая архитектура подключаемых модулей, расширяющих базовые возможности фреймворка;
- огромное сетевое сообщество пользователей;
- великолепная документация, доступная по адресу `http://api.jquery.com`;
- полезные расширения, такие как jQuery UI, предоставляющие дополнительную функциональность.

## История возникновения jQuery

Библиотека jQuery, детище Джона Резига, была впервые представлена общественности на компьютерной конференции BarCamp в Нью-Йорке в начале 2006 года (более подробную информацию о международной сети конференций BarCamp см. на сайте `http://barcamp.org`). Как отметил на своем сайте сам Резиг, к созданию jQuery его подтолкнули неудовлетворенность существовавшими на то время библиотеками и осознание того факта, что их можно значительно улучшить, уменьшив объем "синтаксической шелухи" и введя специфические элементы управления для часто выполняемых операций (`http://ejohn.org/blog/selectors-in-javascript/`).

Библиотека jQuery сразу же завоевала признание сообщества разработчиков и быстро распространилась среди его членов. В процесс усовершенствования библиотеки включились другие разработчики, итогом чего стал выпуск первой устойчивой версии 1.0, состоявшийся 26 августа 2006 года.

С тех пор библиотека jQuery непрерывно улучшается (к моменту написания данной книги была выпущена версия jQuery 1.4.2), и в настоящее время она снабжена множеством дополнений (подключаемых модулей), предложенных сообществом разработчиков. **Подключаемый модуль**, или **плагин** (plug-in), — это расширение, не входящее в состав основной библиотеки. Подробнее о подключаемых модулях jQuery вы узнаете в главе 10.

# Настройка тестовой среды

Поскольку лучший способ освоения нового языка — практические занятия, вам понадобится тестовая среда, предназначенная для выполнения учебных упраж-

нений с использованием jQuery. К счастью, настройка этой тестовой среды представляет собой простой двухшаговый процесс, включающий установку Firefox и Firebug.

На протяжении всей книги будет предполагаться, что вы используете браузер Firefox с подключаемым модулем Firebug, поскольку они предоставляют в ваше распоряжение превосходную тестовую консоль JavaScript.

## Установка Firefox

Чтобы установить на своем компьютере Firefox, посетите сайт `http://firefox.com` и запустите последнюю версию Firefox (версия 3.6 на момент написания книги). После завершения программы установки (`Firefox Setup x.x.x` для Windows или `Firefox x.x.x` для Mac) можно запустить Firefox.

## Установка Firebug

Чтобы установить Firebug, посетите, используя браузер Firefox, сайт `http://getfirebug.com` и щелкните на кнопке Install Firebug For Firefox для открытия диалогового окна программы установки (рис. 1.1). Щелкните на кнопке Установить сейчас (Install Now) и дождитесь окончания процесса установки. После этого перезапустите Firefox.

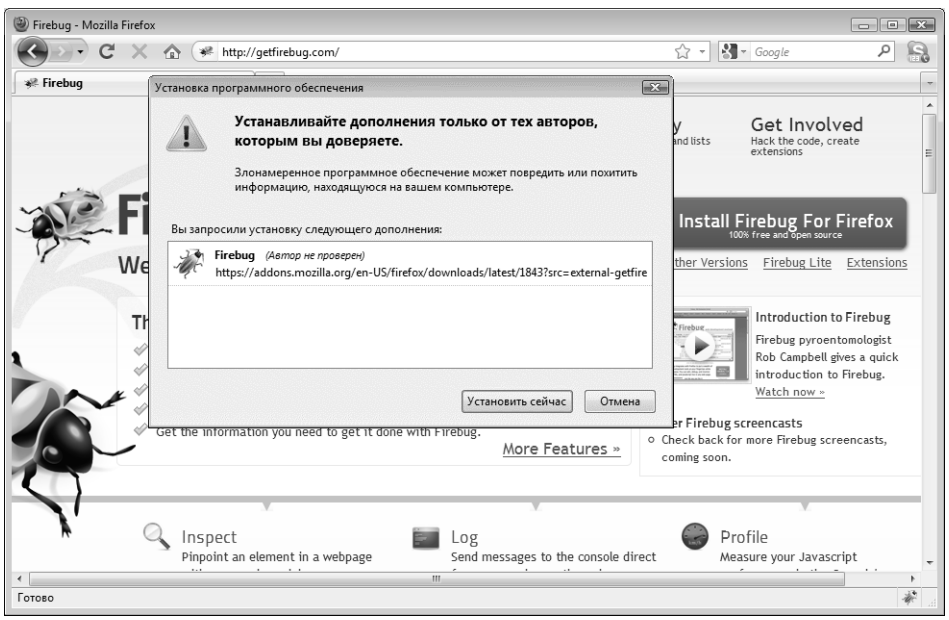



**Рис. 1.1.** Диалоговое окно программы установки Firebug

Перезапустив Firefox, вы увидите, что в правой части строки состояния появился новый значок с изображением жучка. Щелчок на этом значке выведет на экран строку элементов управления Firebug, начинающуюся с элемента Консоль (Console) (рис. 1.2).


**Рис. 1.2.** Так выглядит открытая панель консоли Firebug

**Примечание.** Спектр полезных функций Firebug не ограничивается отладкой JavaScript. Это дополнение окажет неоценимую помощь любому веб-разработчику. Более подробные сведения о Firebug можно получить по адресу `http://getfirebug.com`.

## Настройка локальной тестовой среды

Настраивать локальную тестовую среду, позволяющую выполнять приведенные в книге учебные упражнения, вовсе не обязательно, но если вы это сделаете, то поступите весьма мудро. Локальное тестирование делает процесс разработки более быстрым и безопасным и обычно выполняется проще, чем в случае использования удаленного сервера для этих целей.

### Установка XAMPP

Чтобы быстро и без особых усилий развернуть локальную среду разработки на своем компьютере, загрузите и установите сервер XAMPP[1], придерживаясь следующей процедуры.

1. Посетите сайт `http://www.apachefriends.org/en/xamp.html` и загрузите самую последнюю версию XAMPP, соответствующую вашей операционной системе.
2. Откройте загруженный файл. На компьютере Windows запустите EXE-файл, выберите папку и установите пакет. На компьютере Mac смонтируйте файл DMG и перетащите папку `xampp` в свою папку `Applications`.
3. Откройте панель управления XAMPP (XAMPP Control Panel), запустив файл `xampp-control.exe`, находящийся в папке `xampp`, и запустите сервер Apache.
4. Дабы удостовериться в том, что XAMPP работает, перейдите в браузере на страницу `http://localhost/`. Если сервер функционирует нормально, на экране отобразится домашняя страница XAMPP.

[1] Легко устанавливаемая кроссплатформенная сборка веб-сервера, включающая в себя собственно веб-сервер Apache, СУБД MySQL, PHP, Perl, FTP-сервер, диспетчерскую программу phpMyAdmin и многое другое. — *Примеч. ред.*

Помимо версий XAMPP для Windows и Mac существуют также дистрибутивные пакеты для операционных систем Linux и Solaris. Установка XAMPP в каждой из перечисленных операционных систем имеет свои особенности. Когда будете устанавливать и запускать локальную тестовую среду на своем компьютере, обязательно обратитесь к справочному руководству для получения дополнительной информации.

# Подключение jQuery к веб-страницам

Чтобы jQuery можно было использовать в проекте, библиотека должна быть загружена в HTML-документ, что позволит сценариям обращаться к библиотечным методам. Если библиотека не была предварительно загружена, то наличие любого сценария, в котором используется синтаксис jQuery, вероятнее всего, приведет к появлению ошибок JavaScript. К счастью, загрузить jQuery не составляет труда, и для этого существует ряд возможностей, которыми могут воспользоваться разработчики.

## Подключение загруженной копии библиотеки jQuery

Один из способов организации доступа к библиотеке jQuery состоит в том, чтобы сохранить ее копию в файловой структуре проекта и подключить ее, как любой другой файл JavaScript.

```
<script type="text/javascript" src="js/jquery-1.4.2.min.js">
/script>
```

## Подключение копии библиотеки jQuery, хранящейся на удаленном хосте

Еще один возможный вариант — включение в проект копии библиотеки jQuery, хранящейся на сайте Google Code. Это делается в надежде на то, что к моменту посещения вашего сайта пользователи уже будут иметь копию библиотеки, кешированную с какого-либо другого веб-сайта, который в свое время загрузил эту библиотеку с Google Code, что обеспечит сокращение времени загрузки страниц для пользователей вашего сайта.

Копия библиотеки, хранящаяся на удаленном сервере, включается в проект аналогично загруженной копии.

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
```

## Использование Google Libraries API

На сайте Google Code предлагается еще один вариант загрузки jQuery — с помощью Libraries API (`http://code.google.com/apis/ajaxlibs`). В соответствии с определением, которое дает Google, “Libraries API представляет собой сеть распространения содержимого и архитектуру для загрузки большинства популярных библиотек JavaScript с открытым исходным кодом”.

Использование Libraries API не вызывает никаких трудностей, и именно этот метод мы будем применять на протяжении всей книги. Чтобы подключить jQuery к своему веб-сайту с помощью Libraries API, используйте следующий фрагмент кода.

```
<script type="text/javascript"
    src="http://www.google.com/jsapi"></script>
```

```
<script type="text/javascript">
    google.load("jquery", "1.4.2");
</script>
```

# Создание тестового файла

Настроив тестовую среду, создайте в папке `htdocs` установленной на вашем компьютере сборки XAMPP подпапку `testing`, а в ней — новый файл с именем `index.html`. Выбрав любой удобный для вас текстовый редактор, введите в этот файл показанный ниже HTML-код.

```
<!DOCTYPE html>
<html>
<head>
    <title>Тестирование jQuery</title>
</head>
<body>
    <p>Привет всем!</p>
    <p class="foo">Другой абзац, но уже с классом.</p>
    <p><span>Это контейнер SPAN внутри абзаца.</span></p>
    <p id="bar">Абзац с идентификатором.
        <span class="foo">А это контейнер SPAN внутри него.</span>
    </p>

    <script type="text/javascript"
        src="http://www.google.com/jsapi"></script>
    <script type="text/javascript">
        google.load("jquery", "1.4.2");
    </script>
</body>
</html>
```

**Примечание.** Чтобы избежать блокирования загрузки других элементов страницы, например изображений, вставляйте код, загружающий библиотеку, непосредственно перед закрывающим дескриптором тела документа (`</body>`). Тем самым вы также предотвратите выполнение сценария JavaScript до полной загрузки всех элементов страницы, что могло бы привести к непредвиденному поведению программы или появлению ошибок JavaScript.

Сохраните файл и выполните в Firefox переход по адресу: `http://localhost/testing` (рис. 1.3).

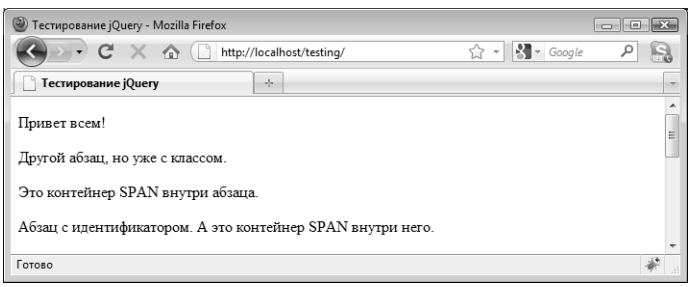



**Рис. 1.3.** Тестовый файл, загруженный в Firefox

Мы будем постоянно использовать этот файл при изучении основных операций jQuery.

## Функция `jQuery()`

В основе jQuery лежит функция `jQuery()`. Эта функция является ключевым элементом библиотеки и так или иначе вызывается всеми методами jQuery. В большинстве реализаций jQuery для вызова функции `jQuery()` используется ее более короткий псевдоним — `$()`, что позволяет уменьшить размер кода.

Мы не будем углубляться в теоретические аспекты программирования и пытаться объяснить все, что происходит в недрах этой функции. Достаточно лишь сказать, что данная функция создает объект jQuery и трактует переданное ей выражение как свои параметры. После этого она определяет, какие ответные действия следует выполнить, и соответствующим образом модифицирует себя.

---

**Предупреждение.** Функция `$()` используется также в ряде других библиотек JavaScript, поэтому в случае одновременного использования нескольких библиотек могут возникать конфликты. Для исправления подобных ситуаций в jQuery предусмотрена функция `jQuery.noConflict()`. Более подробную информации об этой функции можно получить по следующему адресу:

`http://docs.jquery.com/Core/jQuery.noConflict`

---

## Выбор DOM-элементов с использованием синтаксиса CSS

Все, что происходит в jQuery, так или иначе связано с ее невероятно мощным селекторным движком. Далее рассмотрены различные методы, позволяющие выбирать элементы из объектной модели документа (DOM) с помощью jQuery.

---

**Примечание.** DOM — это коллекция объектов и узлов, предоставляющая программный интерфейс для доступа к HTML-, XHTML- или XML-документам. Эта модель не зависит ни от платформы, ни от языка, откуда следует, что для доступа к информации DOM и ее изменения разработчики могут использовать множество языков программирования (в частности, JavaScript) на самых различных платформах (например, в браузерах), не заботясь о проблемах совместимости.

---

Одной из наиболее сильных и привлекательных черт jQuery является та легкость, с которой разработчик может выбирать элементы внутри DOM. Использование селекторов псевдоклассов CSS[2] необычайно расширяет возможности jQuery. Синтаксис псевдоклассов CSS позволяет разработчику манипулировать конкретными экземплярами элементов в HTML-документе. Особенно просто освоить эту методику будет тем, кто уже работал с CSS, поскольку им придется иметь дело практически с тем же синтаксисом. По сути, не выходя за рамки синтаксиса CSS, можно использовать следующие средства для выбора элементов:

- базовые селекторы;
- иерархические селекторы;
- фильтры:
  - базовые фильтры;
  - фильтры содержимого;
  - фильтры видимости;
  - фильтры атрибутов;
  - фильтры элементов-потомков;
- фильтры форм.

[2] `http://www.w3schools.com/CSS/css_pseudo_classes.asp`.

## Базовые селекторы

Базовые селекторы обеспечивают выбор элементов по типу дескриптора, имени класса, уникальному идентификатору (ID) или любой их комбинации. Перейдите в браузере Firefox по адресу `http://localhost/testing/`, запустите Firebug и щелкните на вкладке Консоль (Console) (рис. 1.4). Если панель консоли отключена, щелкните на значке с изображением стрелки на вкладке Консоль и выберите в открывшемся меню пункт Панель включена (Enabled). Эту консоль вы будет использовать для выполнения всех примеров, приведенных в этой главе.

**Примечание.** Читатели, знакомые с CSS, могут пропустить этот раздел, поскольку поведение описанных ниже селекторов совпадает с поведением их CSS-аналогов.

### Выбор элементов по типу дескриптора

Для выбора элементов по типу дескриптора достаточно указать в качестве селектора имя этого дескриптора (например, `p`, `div` или `span`):

```
элемент
```

Чтобы выбрать все абзацы (<p>) в тестовом документе, введите в командной строке, расположенной в нижней части консоли, следующую команду[3]:

```
$("p");
```

Для выполнения команды необходимо нажать клавишу <Enter>. В консоли отобразится следующий результат (рис. 1.4)[4].

```
>>> $("p");
[ p, p.foo, p, p#bar ]
```

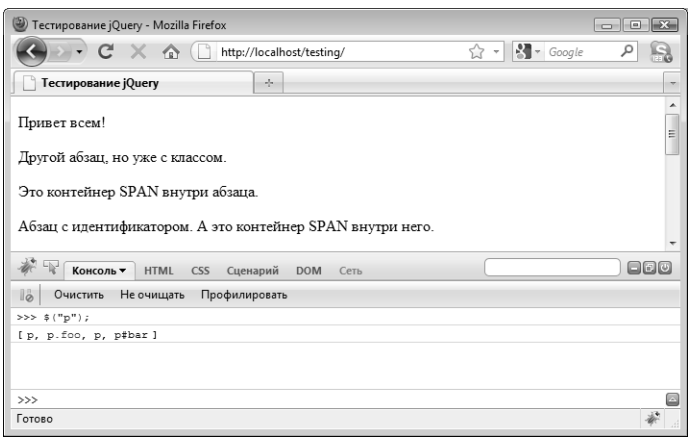



**Рис. 1.4.** Вид консоли Firebug после выполнения команды

[3] Вместо символа кавычек (") можно использовать символ апострофа (`), что избавит от необходимости лишний раз нажимать клавишу <SHIFT>. — *Примеч. ред.*

[4] При наведении указателя мыши на любой из элементов результирующего набора, выведенного в окне консоли, этот элемент выделяется цветовой подсветкой в окне браузера. — *Примеч. ред.*

В первой строке отображается выполняемая команда, а во второй — возвращенный селектором результат. Наш тестовый документ содержит четыре абзаца. Два из них не имеют ни класса, ни идентификатора (ID), третьему присвоен класс `foo`, а четвертому — идентификатор `bar` (изучением соответствующего синтаксиса мы займемся в следующих разделах). При передаче функции `jQuery()` имени дескриптора она находит все вхождения данного дескриптора на странице и добавляет их в объект jQuery.

### Выбор дескрипторов по имени класса

Столь же быстро, как и по типу дескриптора, можно выбирать элементы по присвоенному дескрипторам классу или классам. Для этого надо указать имя класса с точкой (`.`) перед ним:

```
.класс
```

Выбор всех элементов с классом `foo` осуществляется выполнением следующей команды:

```
$(".foo");
```

В консоли отобразится следующий результат.

```
>>> $(".foo");
[ p.foo, span.foo ]
```

Селектор возвратил как дескриптор `p`, так и дескриптор `span`, поскольку оба они имеют класс `foo`.

### Выбор элементов по идентификатору

Для выбора элемента по его идентификатору (атрибуту `id` в синтаксисе CSS) необходимо указать в качестве селектора значение `id`, предварив его символом "решетки" (`#`):

```
#id
```

Найдем все элементы с идентификатором `bar` с помощью следующей команды:

```
$("#bar");
```

Полученный результат согласуется с тем, что в нашем документе присутствует только один элемент с идентификатором `bar`, а именно — `p`.

```
>>> $("#bar");
[ p#bar ]
```

### Уточненный выбор элементов с помощью комбинированных селекторов

Возможны ситуации, когда требуется выделить лишь дескрипторы определенного типа с заданным классом, что легко сделать, использовав в селекторе сочетание типа дескриптора и класса.

Введите в консоли следующую команду, отбирающую лишь абзацы с классом `foo`:

```
$("p.foo");
```

Как видно из приведенного ниже результата, элемент `span` не был отобран, хотя и имеет класс `foo`.

```
>>> $("p.foo");
[p.foo]
```

### Групповые селекторы

В тех случаях, когда необходимо получить доступ сразу к нескольким наборам элементов, используются групповые селекторы. Например, чтобы отобрать абзацы класса `foo` и все элементы, которые имеют идентификатор `bar`, используйте следующую команду:

```
$("p.foo,#bar");
```

Эта команда возвратит элементы, соответствующие *хотя бы одному* из указанных в ней селекторов.

```
>>> $("p.foo,#bar");
[ p.foo, p#bar ]
```

## *Иерархические селекторы*

Иногда возможностей выбора объектов DOM на основании типа элемента, класса или идентификатора оказывается недостаточно. В частности, вам может потребоваться доступ к элементам, которые содержатся внутри заданного элемента, соседствуют с ним или располагаются после него в другом месте документа. Например, такая задача возникает, если необходимо удалить определенный активный класс из всех пунктов меню, кроме одного, на котором был выполнен щелчок, извлечь все списочные элементы из выбранного неупорядоченного списка или изменить атрибуты элемента-оболочки при выборе элемента формы.

### Выбор элементов-потомков

Выбор **потомков**, т.е. элементов, которые содержатся внутри других элементов, называемых **предками**, осуществляется с помощью конструкции, включающей в себя селектор предка (ancestor), пробел и селектор потомка (descendant):

```
предок потомок
```

Чтобы выбрать все элементы `span`, являющиеся потомками элемента `body`, выполните в консоли Firebug следующую команду:

```
$("body span");
```

Эта команда возвратит все элементы `span`, содержащиеся в теле документа, хотя при этом они находятся еще и внутри дескрипторов абзацев.

```
>>> $("body span");
[ span, span.foo ]
```

### Выбор дочерних элементов

Селектор **дочерних** элементов осуществляет более специфический отбор элементов по сравнению с селектором потомков. Он отбирает лишь элементы, являющиеся непосредственными потомками элементов-предков (родителей). Для выбора дочернего элемента используется конструкция, включающая в себя селектор родительского элемента (parent), символ “больше” (>) и селектор дочернего элемента (child):

```
родительский>дочерний
```

Попробуем выбрать в тестовом файле все элементы `span`, являющиеся непосредственными потомками элемента `body`:

```
$("body>span");
```

Поскольку не существует ни одного элемента `span`, который содержался бы непосредственно в элементе `body`, вывод окажется пустым.

```
>>> $("body>span");
[ ]
```

Отберем все элементы `span`, являющиеся непосредственными потомками элементов `p`:

```
$("p>span");
```

Соответствующий вывод будет выглядеть следующим образом.

```
>>> $("p>span");
[ span, span.foo ]
```

### Выбор следующего элемента

Иногда в сценарии может потребоваться выбрать следующий элемент в DOM-структуре. Конструкция такого селектора включает в себя указание элемента, определенного как *начальный* (здесь можно использовать любой селекторный шаблон), знак "плюс" (+) и селектор, выбирающий элемент, определенный как *следующий*:

```
начальный+следующий
```

Проверьте, как это работает, введя в консоли следующую команду:

```
$(".foo+p");
```

Так как в нашем тестовом документе есть только один элемент с классом `foo`, будет возвращен лишь один элемент абзаца.

```
>>> $(".foo+p");
[ p ]
```

Далее создайте более общий запрос и выберите абзацы, каждый из которых следует непосредственно за каким-либо абзацем:

```
$('p+p');
```

Наша разметка содержит четыре абзаца, и для каждого из них, кроме последнего, имеется следующий абзац, поэтому результирующий набор будет содержать три элемента.

```
>>> $("p+p");
[ p.foo, p, p#bar
```

В результирующий набор вошли второй, третий и четвертый абзацы HTML-разметки.

### Выбор сестринских элементов

Все элементы, которые содержатся внутри одного и того же элемента (имеют общего непосредственного предка), являются **сестринскими** по отношению друг к другу. Отбор сестринских элементов осуществляется точно так же, как и отбор следующих элементов, за исключением того, что данный селектор будет отыскивать *все*

сестринские элементы, располагающиеся после начального, а не только *следующий*, т.е. тот, который располагается непосредственно вслед за начальным.

Для выбора сестринских элементов необходимо указать селектор начального элемента, символ "тильды" (~) и селектор сестринских элементов:

```
начальный~сестринский
```

Чтобы найти все элементы, являющиеся сестринскими по отношению к абзацу с классом `foo`, выполните в консоли следующую команду:

```
$(".foo~p");
```

Результат будет таким.

```
>>> $(".foo~p");
[ p, p#bar
```

## Базовые фильтры

Еще одним эффективным методом доступа к DOM-элементам является использование фильтров. В этом случае поиск элементов осуществляется не по их типу, классу или идентификатору, а по их позиции, текущему состоянию или другим переменным.

Базовый синтаксис фильтра включает в себя двоеточие, вслед за которым указывается фильтр.

```
:фильтр
```

Некоторым фильтрам можно передавать параметры, указывая их в скобках.

```
:фильтр(параметр)
```

Ниже рассмотрены наиболее распространенные фильтры.

**Примечание.** Чтобы вы могли быстрее приступить к фактической разработке, будут рассмотрены не все доступные фильтры. Полный список фильтров см. в документации jQuery.

### Выбор первого или последнего элемента

Фильтры часто используют для того, чтобы определить, является ли некоторый элемент первым или последним в наборе элементов. С помощью фильтров эта задача решается очень просто. Для этого достаточно присоединить к любому селектору соответственно фильтр `:first` или `:last`.

```
$("p:last");
```

Выполнение этой команды в консоли Firebug приведет к следующему результату.

```
>>> $("p:last");
[ p#bar ]
```

### Выбор элементов, не соответствующих селектору

В тех случаях, когда требуется найти все элементы, *не соответствующие* определенному селектору, проще всего это сделать с помощью фильтра `:not()`. Если присоединить этот фильтр к селектору и указать в нем другой селектор в качестве параметра, то в результирующий набор попадут лишь те элементы, которые соответствуют исходному селектору, но не соответствуют селектору, переданному фильтру `:not()` в виде параметра.

Например, использование команды `$("p:not(.foo)");` даст следующий результат.

```
>>> $("p:not(.foo)");
[ p, p, p#bar ]
```

### Выбор элементов по признаку четности

Фильтры `:even` и `:odd`, имеющие столь же простой синтаксис, что и фильтры `:first` и `:last`, выбирая из результирующего набора, как несложно догадаться, соответственно четные (even) и нечетные (odd) элементы[5]. Например, выполнение команды `$("p:odd");` приведет к следующему результату.

```
>>> $("p:odd");
[ p.foo, p#bar ]
```

### Выбор элементов по индексу

Для выбора элемента по его индексу используется фильтр `:eq()`, которому в качестве параметра передается индекс требуемого элемента. Например, использование команды `$("p:eq(3)");` даст следующий результат.

```
>>> $("p:eq(3)");
[ p#bar ]
```

**Примечание.** Индекс элемента соответствует его порядковому номеру в наборе. Как это обычно принято в программировании, индексация начинается с нуля, поэтому индекс первого элемента — 0, второго — 1 и т.д.

## *Фильтры содержимого*

Имеются также фильтры, позволяющие отбирать элементы на основании их содержимого, причем с возможностью разграничения текстового и элементного содержимого.

### Выбор элементов, содержащих заданный текст

Для выбора элементов, содержащих определенный текст, используется фильтр `:contains()`, которому искомый текст передается в качестве параметра.

```
$("p:contains(Другой)");
```

Выполнение этой команды в консоли приведет к следующему результату.

```
>>> $("p:contains(Другой)");
[ p.foo ]
```

**Примечание.** Фильтр `:contains()` чувствителен к регистру, т.е. при поиске текста учитывается регистр букв. Версия этого фильтра, нечувствительная к регистру, была предложена одним из членов сообщества разработчиков в его комментариях к статье о фильтре `:contains` в документации API. Для получения более подробной информации обратитесь по адресу `http://api.jquery.com/contains-selector`.

[5] Отбор осуществляется по индексам, отсчет которых начинается с 0. — *Примеч. ред.*

### Выбор элементов, содержащих указанный элемент

Если необходимо найти элементы, содержащие указанный элемент, следует использовать фильтр `:has()`. Этот фильтр работает аналогично фильтру `:contains()`, но принимает в качестве параметра не текст, а имя элемента.

```
$("p:has(span)");
```

Выполнение этой команды в консоли приведет к следующему результату.

```
>>> $("p:has(span)");
[ p, p#bar
```

### Выбор пустых элементов

Для выбора пустых элементов (т.е. элементов, которые не содержат ни других элементов, ни текстовых узлов) используется фильтр `:empty`.

В HTML-примере, который мы используем, все имеющиеся пустые элементы являются невидимыми. Выберите их, задав поиск пустых элементов:

```
$(":empty");
```

Вы получите следующий результат.

```
>>> $(":empty");
[ script jsapi, script jquery.min.js, div#_firebugConsole ]
```

Второй дескриптор `script` и блок `div` сгенерированы динамически. Дескриптор `script` происходит от экземпляра библиотеки jQuery, загруженного Google JSAPI, а дескриптор `div` — от Firebug.

### Выбор родительских элементов

Противоположностью фильтра `:empty` является фильтр `:parent`, отбирающий лишь элементы, являющиеся родительскими по отношению к каким-либо другим (дочерним) элементам или текстовым узлам.

Выберите все абзацы, являющиеся родительскими по отношению к любым другим элементам, воспользовавшись следующей командой:

```
$("p:parent");
```

Поскольку во всех абзацах нашего тестового HTML-документа содержится текст (или, в некоторых случаях, другие элементы), все они попадут в результирующий набор.

```
>>> $("p:parent");
[ p, p.foo, p, p#bar ]
```

### *Фильтры видимости*

Фильтры видимости `:hidden` и `:visible` служат для нахождения соответственно скрытых и видимых элементов. Выберите все видимые абзацы с помощью показанной ниже команды:

```
$("p:visible");
```

Поскольку ни один из элементов в нашем HTML-примере не был скрыт, будет получен следующий результат.

```
>>> $("p:visible");
[ p, p.foo, p, p#bar ]
```

### *Фильтры атрибутов*

Отличные возможности для выбора элементов предоставляют их атрибуты. **Атрибут** — это любой объект, который принадлежит элементу и дополнительно определяет его (в качестве примера можно привести такие атрибуты, как `class`, `href`, `ID` или `title`). В приведенных ниже примерах вы будете использовать атрибут `class`.

**Примечание.** Имейте в виду, что для повышения быстродействия (а также для соблюдения корректного стиля программирования) в реальных сценариях везде, где только возможно, следует использовать селекторы идентификаторов (`#id`) и классов (`.class`); приводимые ниже примеры рассматриваются лишь для того, чтобы продемонстрировать возможности фильтров.

#### Выбор элементов по значению атрибута

Для нахождения элементов с заданными значениями атрибутов используется пара "атрибут-значение", заключенная в квадратные скобки (`[]`).

```
[атрибут=значение]
```

Для выбора всех элементов с атрибутом `.class`, равным `foo`, выполните в консоли следующую команду:

```
$("[class=foo]");
```

Возвращенный результат будет иметь следующий вид.

```
>>> $("[class=foo]");
[ p.foo, span.foo ]
```

#### Выбор элементов, не имеющих заданного атрибута или имеющих другое его значение

Для решения противоположной задачи — выбора элементов, которые не соответствуют заданной паре "атрибут-значение", поместите перед знаком равенства, стоящим между атрибутом и его значением, восклицательный знак (`!`).

```
[атрибут!=значение]
```

Выберите все абзацы, которым не был присвоен класс `foo`, выполнив следующую команду:

```
$("p[class!=foo]");
```

Результат будет таким.

```
>>> $("p[class!=foo]");
[ p, p, p#bar ]
```

### *Фильтры элементов-потомков*

Фильтры потомков предлагают альтернативу использованию фильтров `:even()`, `:odd()` и `:eq()`. Основное различие между ними состоит в том, что в этом наборе фильтров индексация начинается с `1`, а не с `0`, как, например, в фильтре `:eq()`.

#### Выбор параметров по признаку четности, индексу или уравнению

Один из самых универсальных фильтров, `nth-child`, допускает четыре различных варианта передачи параметров при выборе элементов: `even`, `odd`, `index` и `equation`.

Подобно другим фильтрам элементов-потомков, индексация в этом фильтре начинается с 1, а не с 0, поэтому индексом первого элемента является 1, второго — 2 и т.д.

Когда мы использовали фильтр :odd (нечетные элементы), результирующий набор содержал абзацы с классом foo и идентификатором foo. Чтобы проследить за отличиями в работе фильтров, выберите нечетные абзацы с помощью фильтра :nth-child(), воспользовавшись для этого следующей командой:

```
$("p:nth-child(odd)");
```

В консоли отобразится следующий результат.

```
>>> $("p:nth-child(odd)");
[ p, p ]
```

Хотя этот результат и может несколько озадачить, расхождения между обоими указанными случаями являются следствием различий в способах отсчета индексов элементов.

### Выбор первого или последнего дочернего элемента

Фильтры :first-child и :last-child очень напоминают фильтры :first и :last, однако отличаются от них тем, что могут возвращать целый набор соответствующих элементов. Например, чтобы найти последний элемент span, являющийся дочерним по отношению к элементу p, можно использовать следующую команду:

```
>>> $("p span:last");
```

Результат будет таким.

```
>>> $("p span:last");
[ span.foo ]
```

Но если необходимо найти *все* элементы span, являющиеся последними дочерними элементами по отношению к элементу p, то следует использовать фильтр :last-child.

```
$("p span:last-child");
```

Этот фильтр применяется к каждому родительскому элементу, а не к DOM в целом, что и обусловливает отличие возвращенного им результата.

```
>>> $("p span:last-child");
[ span, span.foo ]
```

## *Фильтры форм*

Доминирующая роль форм, которые в наши дни используются практически на всех веб-сайтах, побудила к созданию ряда фильтров, специально предназначенных для работы с ними.

Поскольку в нашем тестовом HTML-примере элементы формы отсутствуют, для выполнения следующих упражнений нам потребуется дополнить файл новой разметкой.

Добавьте в файл index.html следующий HTML-код, поместив его между последним дескриптором p и первым дескриптором script.

```
<form action="#" method="post">
    <fieldset>
        <legend>Форма для ввода учетных данных</legend>
```

```
        <label for="name">Имя</label><br />
        <input name="name" id="name" type="text" /><br />
        <label for="password">Пароль</label><br />
        <input name="password" id="password"
              type="password" /><br /><br />
        <label>
            <input type="radio" name="loc" />
            Я использую свой компьютер
            </label><br />
        <label>
            <input type="radio" name="loc" checked="checked" />
            Я использую общий компьютер
            </label><br /><br />
            <input type="submit" value="Войти" /><br />
        <label>
            <input type="checkbox" name="notify"
                    disabled="true" />
            Сохранить мои учетные данные на этом компьютере
        </label><br />
    </fieldset>
</form>
```

После сохранения файла и перезагрузки страницы `http://localhost/testing/` в браузере должна отобразиться следующая тестовая форма (рис. 1.5).

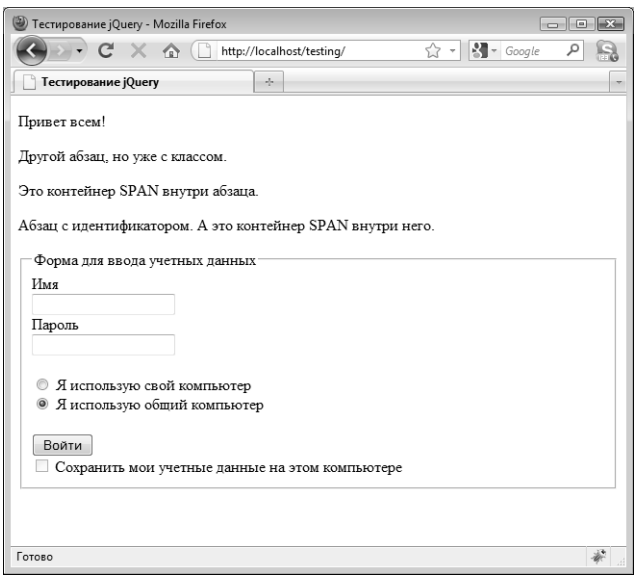



**Рис. 1.5.** Вид формы после изменения содержимого файла `index.html`

## Поиск соответствий по типу элемента формы

Наиболее распространенные, специфические для форм фильтры просто находят элементы управления определенного типа, присутствующие в форме. Доступны следующие фильтры: `:button`, `:checkbox`, `:file`, `:image`, `:input`, `:password`, `:radio`, `:submit` и `:text`.

Чтобы выбрать все переключатели, используйте следующую команду:

```
$("input:radio");
```

Результат будет таким.

```
>>> $("input:radio");
[ input on, input on ]
```

Эти фильтры особенно полезны ввиду того, что все доступные типы представляют собой элементы ввода, нахождение которых без использования указанных фильтров было бы немного труднее выполнить.

### Выбор включенных и отключенных элементов формы

Имеются также фильтры `:enabled` и `:disabled`, с помощью которых можно отбирать элементы, находящиеся соответственно во включенном или отключенном состоянии. Чтобы выбрать все отключенные элементы формы, используйте следующий код:

```
$(":disabled");
```

Результат будет таким.

```
>>> $(":disabled");
[ input on ]
```

Флажок Сохранить мои учетные данные на этом компьютере в нашей форме отключен и поэтому был возвращен фильтром `:disabled`.

### Выбор отмеченных или выделенных элементов формы

Флажки и переключатели имеют состояние `checked`, а списки — состояние `selected`. Предусмотрены фильтры, с помощью которых можно отбирать элементы, находящиеся в одном из этих состояний, используя соответственно синтаксис `:checked` и `:selected`.

Чтобы выбрать включенный переключатель в нашем HTML-примере, выполните в консоли следующую команду:

```
$(":checked");
```

Будет возвращен переключатель, который в настоящее время включен.

```
>>> $(":checked");
[ input on ]
```

# Резюме

В этой главе вы узнали о том, что такое jQuery и зачем создавалась эта библиотека, а также изучили основные приемы работы с ней. Кроме того, вы настроили среду разработки, использующую XAMPP, Firefox и дополнение Firebug. К этому моменту вы уже должны чувствовать себя уверенно при выборе элементов DOM с помощью мощного селекторного движка jQuery. Материал главы мог показаться несколько суховатым, однако очень важно, чтобы вы до конца понимали, как работает jQuery, прежде чем мы примемся за создание более сложного кода.

В следующей главе вы изучите способы обхода узлов DOM, основанные на использовании встроенных методов jQuery.

Глава 2

# Распространенные операции и методы jQuery

Теперь, когда вы уже поняли, как работает механизм выбора элементов, можно приступить к изучению основных методов работы с jQuery, упрощающих организацию интерактивного взаимодействия пользователей с веб-страницами. В этой главе вы познакомитесь с наиболее полезными и чаще всего используемыми средствами jQuery.

Эта глава напоминает краткий справочник и может показаться несколько суховатой, но в ваших же интересах тщательно проработать все представленные в ней примеры. Приобретенные при этом знания принципов работы описанных здесь методов составят вам неоценимую службу, когда вы приступите к разработке реального проекта.

## Особенности поведения сценариев jQuery

Одной из наиболее притягательных особенностей библиотеки jQuery является то, что практически любой из ее методов способен объединяться с другими методами в **цепочки**, в которых методы автоматически вызываются один за другим. Тем самым обеспечивается ясность и сжатость кода, что повышает его удобочитаемость и облегчает анализ.

```
$('p')
.addClass('new-class')
.text("Это абзац!")
.appendTo('body');
```

Объединение методов в цепочки возможно благодаря тому, что для каждого из них объект jQuery является как передаваемым, так и получаемым типом. Это позволяет использовать сокращенный синтаксис цепочек, а не вызывать каждый из входящих в цепочку методов по отдельности. Поначалу данная концепция может восприниматься с трудом, но, когда вы поработаете с примерами, ситуация для вас прояснится.

## Методы jQuery

Библиотека jQuery призвана облегчить решение повседневных задач программирования. Если говорить коротко, то упрощение процесса разработки с исполь-

зованием JavaScript достигается за счет предоставления разработчику следующих удобных возможностей:

- выбор DOM-элементов с использованием синтаксиса CSS (это рассматривалось в главе 1);
- простые процедуры прохождения узлов и изменения DOM-структуры;
- несложный синтаксис для обработки событий браузера (например, щелчков на элементах управления или помещения указателя мыши над ними);
- доступ ко всем атрибутам элемента, включая стилевые свойства, и возможность их изменения;
- анимация и другие эффекты;
- простые элементы управления AJAX.

**Примечание.** Приведенный перечень лишь частично отражает особенности и возможности jQuery. По мере чтения книги вы познакомитесь с другими полезными средствами. Как обычно, для получения подробных справочных сведений обращайтесь к документации по адресу `http://api.jquery.com`.

## Обход DOM-элементов

В jQuery **обход** элементов — это просмотр существующего дерева DOM-элементов. По сути, эта операция представляет собой разновидность фильтрации, которая выполняется *после* произведенного перед этим отбора элементов. Такая возможность чрезвычайно удобна для разработчиков, поскольку позволяет выполнить манипуляции с одной частью DOM-структуры, а затем перейти к другой ее части без повторного проведения процедуры поиска с помощью селекторов.

Данное средство значительно облегчает жизнь разработчикам, обеспечивая простой способ доступа к непосредственному окружению элемента, с которым в данный момент выполняются какие-либо манипуляции или который так или иначе используется в сценарии. Необходимость в этом может возникнуть в целом ряде других случаев, например когда в родительские элементы требуется добавить класс, позволяющий отслеживать их активность, или отключить все неактивные элементы формы.

**Примечание.** В примерах этой главы используется тот же тестовый файл, что и в главе 1. Если у вас установлен XAMPP, перейдите в браузере по адресу `http://localhost/testing/` для загрузки этого файла. Убедитесь в том, что консоль Firebug открыта и активна (чтобы освежить свои знания по использованию консоли Firebug, обратитесь к главе 1).

### *.eq()*

Если набор найденных селектором элементов необходимо сократить до одного, идентифицируемого с помощью индекса, можете использовать метод `.eq()`. Этот метод принимает единственный параметр: индекс требуемого элемента. Индексация начинается с `0`.

```
$("p").eq(1);
```

При выполнении этой команды в консоли Firebug будет получен такой результат.

```
>>> $("p").eq(1);
[ p.foo ];
```

В методе `.eq()` предусмотрена дополнительная возможность обратного отсчета элементов — от конца выбранного набора с помощью отрицательных значений индекса (например, при передаче методу значения `-2` он вернет второй с конца элемент набора).

Для выбора того же абзаца, что и в предыдущем примере, но с применением обратного отсчета элементов, выполните следующую команду:

```
$("p").eq(-3);
```

Вы получите тот же результат, что и прежде.

```
>>> $("p").eq(-3);
[ p.foo ]
```

### .filter() и .not()

Если внутри отобранного селектором набора элементов необходимо использовать другой селектор, вам пригодится метод `.filter()`. Этот метод принимает в качестве параметра любой допустимый в функции jQuery селектор, но применяется лишь к поднабору элементов, содержащемуся в объекте jQuery.

Например, чтобы выбрать все абзацы, а затем оставить в наборе лишь те из них, которые имеют класс `.foo`, выполните следующую команду:

```
$("p").filter(".foo");
```

В консоли отобразится такой результат.

```
>>> $("p").filter(".foo");
[ p.foo ]
```

Обратные по отношению к методу `.find()` функции выполняет метод `.not()`, возвращающий из найденного набора лишь те элементы, которые не соответствуют данному селектору. Например, чтобы выбрать все абзацы, а затем ограничить найденный набор абзацами, не имеющими класса `foo`, выполните следующую команду:

```
$("p").not(".foo");,
```

В консоли отобразится такой результат.

```
>>> $("p").not(".foo");
[ p, p, p#bar ]
```

### .first() и .last()

Методы `.first()` и `.last()` идентичны методам `.eq(0)` и `.eq(-1)` соответственно. Чтобы выбрать из набора всех абзацев на странице последний из них, выполните следующую команду:

```
$("p").last();,
```

Результат ее применения будет таким.

```
>>> $("p").last();
[ p#bar ]
```

### .has()

Метод .has() позволяет выбрать элемент, который содержит элементы, соответствующие определенному шаблону. Например, приведенная ниже команда выбирает все абзацы и оставляет в результирующем наборе лишь те из них, которые содержат элемент span.

```
$("p").has("span");
```

Результат ее применения будет таким.

```
>>> $("p").has("span");
[ p, p#bar ]
```

### .is()

Особенностью метода .is() является то, что он *не возвращает* объект jQuery. Он оценивает результирующий набор, не изменяя его, и тем самым идеально подходит для использования в функциях обратного вызова или функциях, выполняемых лишь после успешного выполнения некоторой другой функции или метода.

Ближе с примерами практического использования функции .is() вы познакомитесь далее, а сейчас выберите все абзацы в тестовом документе и определите, имеются ли среди них абзацы с классом foo.

```
$("p").is(".foo");
```

Результат представляет собой булево значение (true или false), которое дает ответ на интересующий вас вопрос.

```
>>> $("p").is(".foo");
true
```

### .slice()

Если необходимо выбрать подмножество элементов на основании их индексов, в игру вступает метод .slice(). Он принимает два аргумента: индекс начального элемента поднабора и необязательный индекс конечного элемента. Если второй параметр не предоставлен, поднабор продолжается до конца исходного набора.

**Примечание.** Элемент, индекс которого передается в качестве второго параметра, не включается в результирующий набор. Если, например, требуется выбрать из исходного набора элементы со второго по четвертый (т.е. элементы с индексами от 1 до 3), то передаваемыми параметрами должны быть 1 и 4.

Кроме того, как и в случае метода .eq(), допускается передача отрицательных значений параметров, причем это касается указания как начала, так и конца подмножества.

Чтобы выбрать все абзацы, а затем ограничить найденный набор вторым и третьим абзацами, выполните следующую команду:

```
$("p").slice(1,3);
```

В консоли отобразится такой результат.

```
>>> $("p").slice(1,3);
[ p.foo, p ]
```

Для выбора двух последних элементов из набора всех абзацев выполните следующую команду:

```
$("p").slice(-2);
```

Результат ее применения будет таким.

```
>>> $("p").slice(-2);
[ p, p#bar ]
```

### .children()

Часто возникает необходимость в дополнительной детализации результирующего набора с целью нахождения непосредственных потомков (дочерних элементов) для каждого из элементов предварительно найденного набора. Эта задача решается с помощью метода `.children()`, принимающего один необязательный параметр: селектор, которому должны соответствовать дочерние элементы.

Чтобы выбрать все абзацы, а затем изменить принцип отбора так, чтобы в результирующий набор попали только дочерние элементы абзацев, выполните следующую команду:

```
$("p").children();
```

В консоли отобразится такой результат.

```
>>> $("p").children();
[ span, span.foo ]
```

Если необходимо уточнить, какие именно дочерние элементы должны отбираться, можно передать методу `.children()` необязательный селектор. Чтобы выбрать все абзацы, а затем найти среди них все дочерние элементы, имеющие класс `foo`, выполните следующую команду:

```
$("p").children(".foo");
```

В консоли отобразится такой результат.

```
>>> $("p").children(".foo");
[ span.foo ]
```

### .closest()

Метод `.closest()` обеспечивает поиск ближайшего элемента-предка (родительского элемента), соответствующего данному селектору, начиная с текущего элемента и перемещаясь вверх по **DOM-дереву**, представляющему порядок вложения элементов (так, элемент `span` вложен в элемент `p`, который вложен в элемент `body`).

Например, для нахождения абзаца, ближайшего к элементу `span` с классом `foo`, выполните в консоли следующую команду:

```
$("span.foo").closest("p");
```

Результат будет таким.

```
>>> $("span.foo").closest("p");
[ p#bar ]
```

### .find()

Как и метод `.children()`, метод `.find()` предназначен для нахождения в текущем наборе тех элементов-потомков, которые соответствуют указанному селектору. Основное различие между методами `.find()` и `.children()` состоит в том, что метод `.children()` осуществляет поиск только дочерних элементов, т.е. непосредственных

потомков, тогда как метод .find() находит все элементы-потомки, независимо от того, на каком уровне вложенности они находятся.

Чтобы убедиться в справедливости сказанного, выберите дескриптор body, а затем найдите содержащиеся внутри него элементы span, выполнив следующую команду:

```
$("body").find("span");
```

В результате будут возвращены оба элемента span.

```
>>> $("body").find("span");
[ span, span.foo ]
```

Но если вы попытаетесь получить то же самое с помощью метода .children(), то возвращенный набор элементов окажется пустым.

```
>>> $("body").children("span");
[ ]
```

### .next(), .nextAll() и .nextUntil()

Для нахождения элементов, следующих непосредственно за каждым из элементов найденного набора, предназначены три полезных метода: .next(), .nextAll() и .nextUntil().

Метод .next() находит для текущего элемента сестринский элемент, который следует непосредственно за ним. Чтобы выбрать абзац с классом foo, а затем перейти к его ближайшему сестринскому элементу, выполните в консоли приведенную ниже команду.

```
$("p.foo").next();
```

Вы получите следующий результат.

```
>>> $("p.foo").next();
[ p ]
```

Методу .next()можно передать селектор, конкретизирующий тип искомого сестринского элемента.

```
$("p.foo").next("#bar");
```

Выполнение этой команды приведет к пустому результирующему набору, так как абзац, следующий за абзацем с классом foo, не снабжен идентификатором bar.

```
>>>$("p.foo").next("#bar");
[ ]
```

Поскольку метод .next() возвращает лишь ближайший сестринский элемент, предусмотрен аналогичный метод .nextAll(), который возвращает для элемента набора все его сестринские элементы, расположенные после него, а не только соседний. Чтобы выбрать все абзацы, расположенные после абзаца с классом foo, выполните следующую команду:

```
$(".foo").nextAll("p");
```

Результат будет таким.

```
>>>$(".foo").nextAll("p");
[ p, p#bar ]
```

**Примечание.** Как и в случае метода `.next()`, задание селектора при вызове метода `.nextAll()` не является обязательным.

Третьим из доступных методов выбора сестринских элементов является метод `.nextUntil()`. Как ясно из самого названия, этот метод возвращает для данного элемента набора все расположенные после него сестринские элементы до тех пор, пока не встретится элемент, соответствующий селектору. Важно отметить, что сам элемент, соответствующий селектору, не будет включен в результирующий набор.

Чтобы в этом убедиться, выберите абзац с классом `foo` и примените метод `.nextUntil()` с селектором `"#bar"`.

```
$(".foo").nextUntil("#bar");
```

В результирующий набор войдет только один абзац, тогда как абзац с идентификатором `bar` не будет включен в него.

```
>>>$(".foo").nextUntil("#bar");
[ p ]
```

Если все же требуется включить в результирующий набор абзац с идентификатором `bar`, то тогда в качестве элемента, ограничивающего диапазон поиска, необходимо указать элемент, непосредственно следующий за абзацем с идентификатором `bar`, в данном случае — элемент `form`. Вновь примените селектор, используя приведенный ниже обновленный код.

```
$(".foo").nextUntil("form");
```

Теперь будут возвращены оба абзаца.

```
>>>$(".foo").nextUntil("form");
[ p, p#bar ]
```

### *.prev(), .prevAll() и .prevUntil()*

Методы `.prev()`, `.prevAll()` и `.prevUntil()` работают аналогично методам `.next()`, `.nextAll()` и `.nextUntil()`, но осуществляют поиск не следующих, а предыдущих сестринских элементов.

```
>>>$("#bar").prev();
[ p ]

>>>$("#bar").prevAll();
[ p, p.foo, p ]

>>>$("#bar").prevUntil(".foo");
[ p ]
```

### *.siblings()*

Для выбора как предшествующих, так и последующих сестринских элементов, используется метод `.siblings()`. Чтобы ввести ограничения на тип возвращаемых элементов, можно задать соответствующий селектор в качестве параметра. Найдите все абзацы, являющиеся сестринскими по отношению к абзацу с идентификатором `bar`, выполнив приведенный ниже код.

```
$("#bar").siblings("p");
```

Результат будет выглядеть следующим образом.

```
>>>$("#bar").siblings("p");
[ p, p.foo, p ]
```

### .parent()

Метод .parent() возвращает набор элементов, являющихся непосредственными предками (родителями) элементов текущего найденного набора. Например, чтобы выбрать все элементы, являющиеся родительскими по отношению к элементам с классом foo, можно использовать следующую команду:

```
$(".foo").parent();
```

Эта команда возвратит следующий результат.

```
>>>$(".foo").parent();
[ body, p#bar ]
```

Чтобы отобрать из всех элементов, являющихся родительскими по отношению к элементам с классом foo, только абзацы, видоизмените код, как показано ниже.

```
$(".foo").parent("p");
```

Это приведет к сужению результирующего набора.

```
>>> $(".foo").parent("p");
[ p#bar ]
```

### .parents() и .parentsUntil()

Метод .parents(), в отличие от метода .parent(), возвращает все элементы-предки и может принимать необязательный селектор для фильтрации результатов.

Для выбора всех элементов-предков флажка на странице образца формы выполните следующую команду:

```
$(":checkbox").parents();
```

Будут найдены все элементы-предки флажка, вплоть до элемента html.

```
>>>$(":checkbox").parents();
[ label, fieldset, form #, body, html ]
```

Чтобы отфильтровать результат и возвратить лишь элемент form, видоизмените код, как показано ниже.

```
$(":checkbox").parents("form");
```

Это приведет к тому, что из всех элементов-предков флажка будет возвращен только элемент form.

```
>>>$(":checkbox").parents("form");
[ form # ]
```

Наконец, для выбора диапазона родительских элементов, ограниченного элементом, соответствующим селектору (аналогично рассмотренным ранее примерам с методами .nextUntil() и .prevUntil()), используется метод .parentsUntil().

```
$(":checkbox").parentsUntil("form");
```

Приведенная выше команда возвращает все элементы-предки флажка, пока не встретится элемент `form`.

```
>>>$(":checkbox").parentsUntil("form");
[ label, fieldset ]
```

### .add()

Метод `.add()` позволяет добавлять в существующий объект jQuery дополнительные элементы с помощью селектора или строки HTML.

Чтобы выбрать все абзацы, а затем добавить контейнер `span` с классом `foo` в объект, используйте приведенный ниже код.

```
$("p").add("span.foo");
```

Вы получите следующий результат.

```
>>>$("p").add("span.foo");
[ p, p.foo, p, p#bar, span.foo ]
```

Кроме того, метод `.add()` позволяет создавать элементы "на лету", например так:

```
$("p").add('<span id="bat">Это новый контейнер SPAN</span>');
```

Результат выполнения этого кода будет таким.

```
>>>$("p").add('<span id="bat">Это новый контейнер SPAN</span>');
[ p, p.foo, p, p#bar, span#bat ]
```

**Примечание.** Заметьте, что в консольном выводе элемент `span#bat` затенен. Это происходит потому, что, хотя данный элемент и существует в объекте jQuery, он еще не присоединен к DOM-структуре и не отображается на странице. Более подробно о добавлении новых элементов в DOM-структуру документа будет рассказано в следующем разделе.

### .andSelf()

В процессе обхода узлов DOM-структуры может возникнуть потребность в сохранении возможности доступа к первоначально выбранному набору элементов. Такую возможность предоставляет метод `.andSelf()`, который позволяет обратиться к исходному набору для присоединения его к вновь полученному.

Например, приведенный ниже код позволяет осуществить предварительный отбор всех абзацев, а затем выполнить поиск всех дочерних контейнеров `span`.

```
$("p").find("span");
```

Этот код возвратит все содержащиеся в документе элементы `span`, но при этом вы потеряете абзацы.

```
>>>$("p").find("span");
[ span, span.foo ]
```

Чтобы отобрать все элементы `span`, но при этом сохранить и абзацы, добавьте в конец этого кода вызов метода `.andSelf()`.

```
$("p").find("span").andSelf();
```

Это даст возможность получить требуемый результат.

```
>>>$("p").find("span").andSelf();
[ p, p.foo, p, span, p#bar, span.foo ]
```

### .contents()

Метод .contents() работает аналогично методу .children(), за исключением того, что дополнительно возвращаются также текстовые узлы, которые представляют собой символьные данные, содержащиеся внутри элемента (фактический текст, отображаемый в элементе).[1]

Чтобы найти все содержимое элемента span, имеющего класс foo, используйте приведенный ниже код.

```
$("span.foo").contents();
```

Результат выполнения этого кода будет такиа.

```
>>>$("span.foo").contents();
[ <TextNode textContent="А это контейнер SPAN внутри него."> ]
```

### .end()

Иногда в процессе работы со сценариями jQuery может возникнуть необходимость возвратиться к последнему набору элементов, сохраненному в jQuery. Именно для этой цели и предусмотрен метод .end(), возвращающий объект jQuery в состояние, в котором он находился непосредственно перед выполнением последней операции фильтрации в текущей цепочке объектов jQuery.

Приведенный ниже код сначала выбирает все абзацы, а затем находит в полученном наборе все элементы span, при этом исходный набор абзацев становится недоступным.

```
>>>$("p").find("span");
[ span, span.foo ]
```

Чтобы возвратиться к предварительно полученному набору абзацев, добавьте вызов метода .end() в конец цепочки.

```
>>>$("p").find("span").end();
[ p, p.foo, p, p#bar ]
```

## Создание и вставка DOM-элементов

Первое, с чего мы начнем изучение средств, позволяющих не просто выбирать элементы DOM-структуры документа, но и вносить в нее фактические изменения, — это создание новых элементов и вставка их в DOM. С выходом jQuery 1.4 решение этой задачи не представляет никакой сложности.

Начиная с этого раздела мы будем использовать более сложные фрагменты кода, в связи с чем вам придется выполнить незначительную дополнительную настройку своей консоли Firebug. В правой части командной строки консоли находится кнопка с изображением стрелки, направленной вверх (рис. 2.1).

Щелкните на этой кнопке для активизации расширенной области тестирования кода, в которой вы сможете вводить команды, занимающие несколько строк, что упростит их чтение и позволит работать с более сложными примерами (рис. 2.2).

[1] http://www.w3.org/TR/DOM-Level-3-Core/core.html#ID-1312295772.

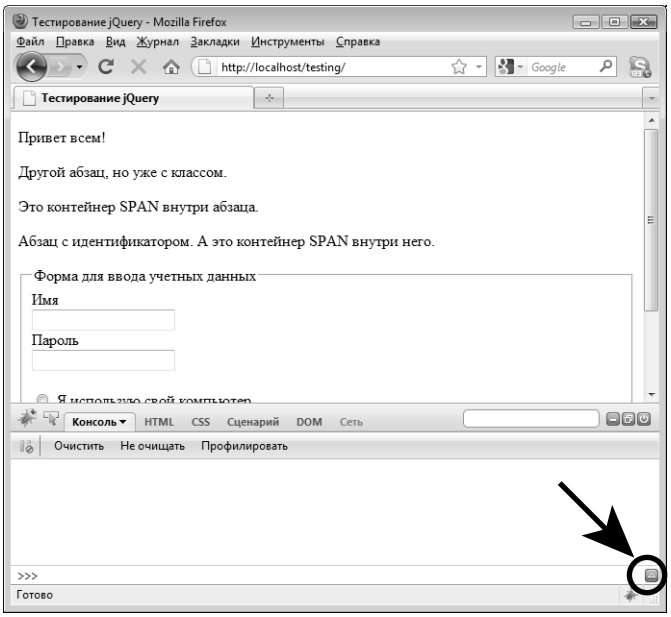



**Рис. 2.1.** Кнопка активизации многострочной командной строки консоли

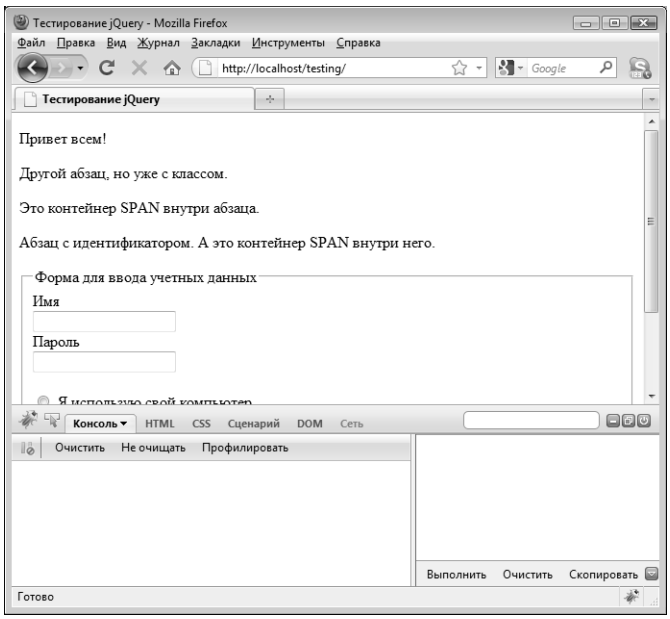



**Рис. 2.2.** Многострочная область тестирования кода (отображается в правой части консоли)

Теперь для выполнения кода, содержащегося в расширенной командной строке, вам придется щелкать на кнопке Выполнить (Run) в нижней части области ввода. Нажатие клавиши <Enter>, как в случае однострочной консоли, будет приводить лишь к переходу на новую строку.

### Создание новых DOM-элементов

Чтобы создать новый DOM-элемент, требуется лишь сообщить jQuery его дескриптор. Например, для создания нового элемента абзаца используется следующий код:

```
$("<p>");
```

Чтобы добавить в элемент атрибуты и текст, достаточно записать их в виде простого HTML-кода.

```
$('<p class="bat">Это новый абзац!</p>');
```

**Примечание.** В предыдущем примере HTML-строка выделена не двойными кавычками, а апострофами. На функционировании jQuery это никак не сказывается; это лишь позволило нам избавиться от необходимости маскировать двойные кавычки, используемые в атрибуте `class` (например, `class=\"bat\"`).

Начиная с версии jQuery 1.4 можно добавлять атрибуты в новый элемент путем передачи второго аргумента с использованием нотации JavaScript Object Notation (JSON)[2].

```
$("<p>", {
    "class":"bat",
    "text":"Это новый абзац!"
});
```

Выполнение приведенного выше фрагмента кода даст следующий результат.

```
>>>$("<p>", { "class":"bat", "text":"This is a new paragraph!" });
[ p.bat ]
```

**Примечание.** Если ограничиться кратким объяснением, то JSON требует указывать пары *ключ–значение*, в которых в кавычки берутся как *ключ*, так и *значение*, а все такие пары отделены друг от друга двоеточиями и совокупно заключены в фигурные скобки, например `{ "key":"value" }` или `{ "key1":"value1", "key2":"value2" }`.

### Вставка новых элементов в DOM

Теперь, когда вы уже понимаете, как создавать новые элементы, можно приступить к вставке их в DOM. Для этого в jQuery предусмотрено несколько методов, которые будут изучены в этом разделе.

Здесь важно отметить, что вносимые таким способом в DOM изменения являются временными в том смысле, что после обновления страницы они будут утеряны и HTML-документ приобретет свой первоначальный вид. Это происходит по той причине, что JavaScript — клиентский язык, откуда следует, что вносимые вами изменения сказываются не на самих файлах на сервере, а лишь на способе их отображения в браузере.

Изменения, вносимые сценариями JavaScript, могут быть сохранены на сервере посредством использования библиотеки AJAX (о чем мы поговорим далее), предоставляющей интерфейс для взаимодействия JavaScript с серверными языками, такими как PHP.

[2] http://ru.wikipedia.org/wiki/JSON.

**Примечание.** Заканчивая работу с примерами в каждом из следующих разделов, всякий раз обновляйте страницу, чтобы каждый новый пример начинался с исходного экземпляра HTML-файла.

### `.append()` и `.prepend()`

Функции `.append()` и `.prepend()` присоединяют к объекту jQuery, из которого они вызываются, элементы, передаваемые им в качестве параметров. Единственное различие между ними состоит в том, что функция `.append()` присоединяет элементы в конец объекта, а `.prepend()` — в начало.

В обоих случаях присоединение содержимого осуществляется *внутри* соответствующих элементов, т.е. если отыскиваются все абзацы на странице и к ним присоединяется новое предложение "Это было добавлено jQuery", то оно будет присоединяться внутри абзаца, до закрывающего дескриптора (`</p>`).

Убедитесь в этом, введя в консоли следующий код:

```
$("p").append(" Это было добавлено jQuery.");
```

В результате выполнения этого кода предложение будет добавлено в конец каждого абзаца перед закрывающим дескриптором. Это довольно очевидно, ибо если бы текст находился за закрывающим дескриптором абзаца, то он перешел бы на следующую строку.

#### Исследование HTML с помощью средства просмотра элементов Firebug

В сказанном выше можно непосредственно убедиться, используя средство просмотра элементов (Element Inspector), предоставляемое консолью Firebug. В верхней части консоли слева имеется кнопка с изображением указателя мыши над прямоугольником (рис. 2.3). Щелкните на ней для активизации указанного средства.

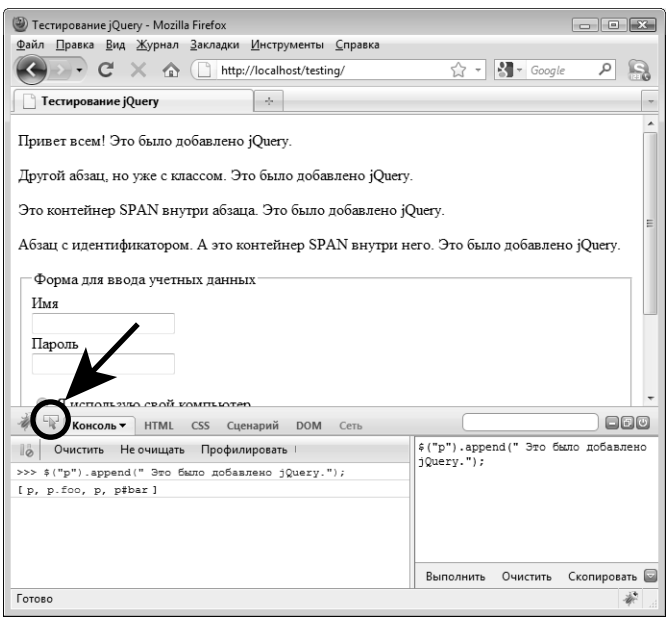



**Рис. 2.3.** Кнопка активизации средства просмотра элементов

После активизации данного средства наведение указателя мыши на любой элемент в окне браузера будет сопровождаться выделением этого элемента голубой рамкой. Наведите указатель на один из абзацев, к которому вы только что присоединили текст, и щелкните на нем. Это активизирует панель HTML с подсвеченным и свернутым текущим элементом. Расположенная рядом с элементом кнопка позволяет развернуть его (рис. 2.4).

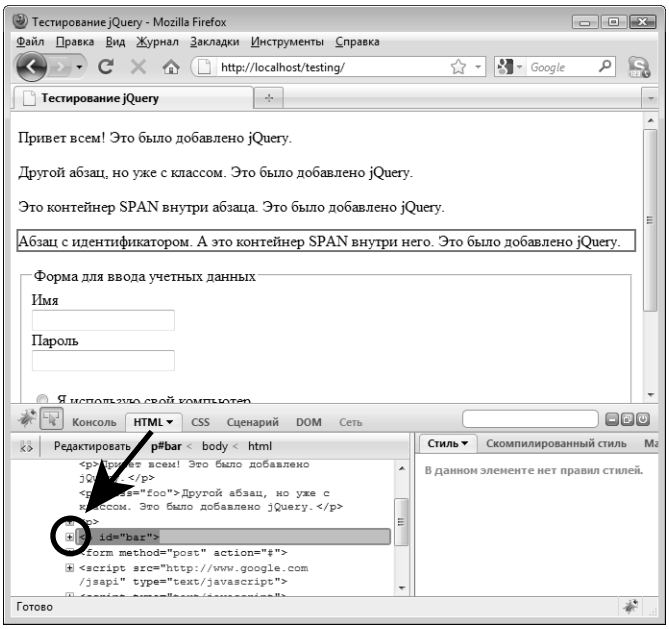



**Рис. 2.4.** Свернутый элемент, отображаемый после выполнения на нем щелчка

Щелкнув на кнопке для развертывания элемента, вы сможете увидеть его содержимое, включая и присоединенный текст, содержащийся внутри абзаца (рис. 2.5).

Вы можете использовать эту методику во всех упражнениях, чтобы контролировать размещение добавляемых в DOM элементов и содержимого.

Функции `.append()` и `.prepend()` также позволяют добавлять в DOM новые элементы. Например, чтобы добавить новый абзац вверху страницы, присоедините новый элемент к элементу `body`, используя приведенный ниже код.

```
var para = $("<p>", {
        "text":"Я в новом абзаце!",
        "css":{"background":"yellow"}
    });
$("body").prepend(para);
```

**Примечание.** В этом примере, прежде чем присоединять новый элемент к телу документа, создается переменная для его хранения. Это сделано для того, чтобы текст сценария стал более понятным. Мы часто будем применять эту методику на протяжении всей книги.

После выполнения приведенного выше кода в верхней части окна браузера появится новый абзац с желтым фоном (рис. 2.6).

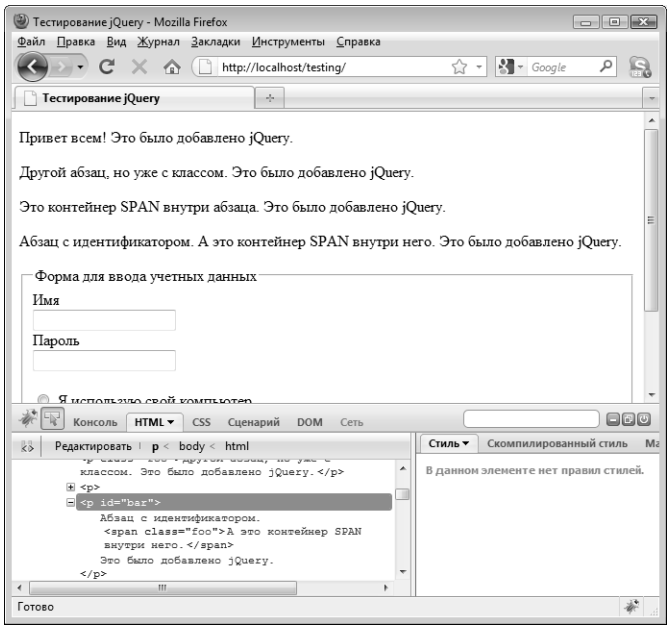



**Рис. 2.5.** Отображение развернутого элемента, включая динамически добавленный текст

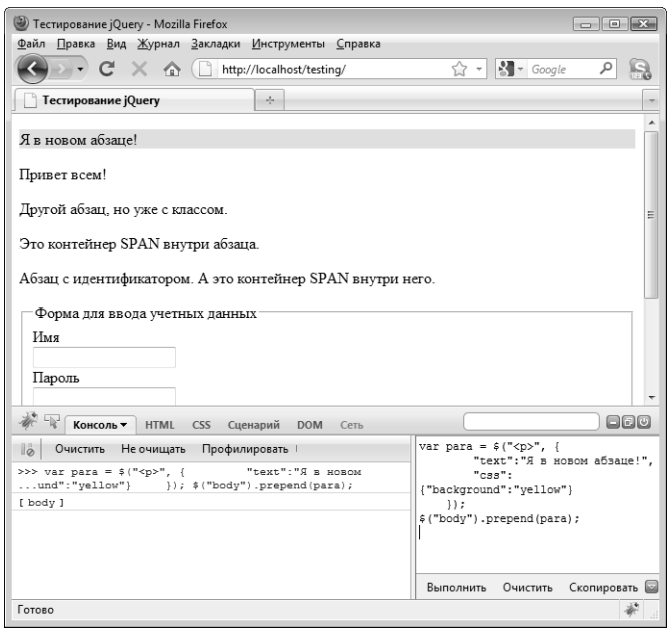



**Рис. 2.6.** Вид нового абзаца после его присоединения к телу документа

### .appendTo() и .prependTo()

В последнем примере вы должны были создать элемент, сохранить его, а затем выбрать элемент, к которому его следует присоединить. Вместе с тем, jQuery предоставляет методы `.appendTo()` и `.prependTo()`, которые, в отличие от двух предыдущих, вызываются из присоединяемого объекта и принимают в качестве параметра селектор элемента, к которому объект присоединяется.

Если в качестве отправной точки взять предыдущий пример, то в случае использования метода `.prependTo()` добавление того же абзаца к телу документа значительно упростится.

```
$("<p>",{
        "text":"Я в новом абзаце!",
        "css":{"background":"yellow"}
    })
    .prependTo("body");
```

Это дает прежний результат, но размер кода при этом заметно уменьшился.

### .after() и .before()

Методы `.after()` и `.before()` аналогичны методам `.append()` и `.prepend()`, но только вместо добавления содержимого в начало или конец элемента они добавляют содержимое перед элементом или после него.

Добавьте новый абзац после абзаца с классом `foo`, используя приведенный ниже фрагмент кода.

```
$("p.foo").after("<p>Новый абзац.</p>");
```

Этот код создаст новый абзац, вставив его непосредственно после абзаца с классом `foo` (рис. 2.7).

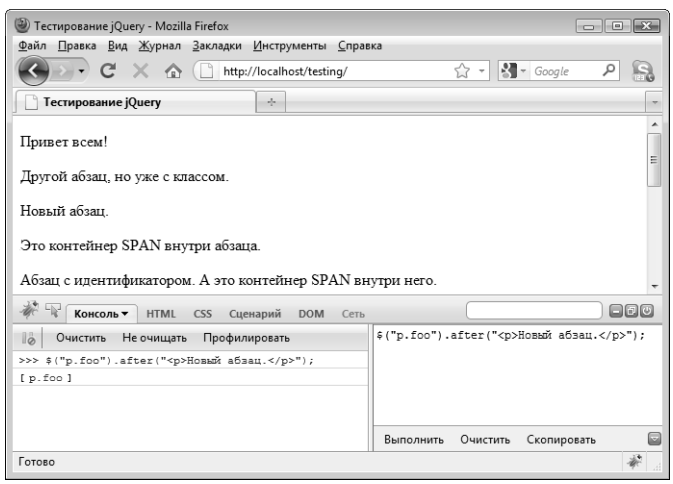



**Рис. 2.7.** Новый абзац, вставленный после абзаца с классом `foo`

### .insertAfter() и .insertBefore()

Подобно тому как методы `.appendTo()` и `.prependTo()` позволяют сократить размер кода, предназначенного для добавления новых элементов в DOM, методы

.insertAfter() и .insertBefore() предлагают аналогичную альтернативу для методов .after() и .before(). Чтобы повторить пример из предыдущего раздела с использованием метода .insertAfter(), измените код, как показано ниже.

```
$("<p>", {
        "text":"Новый абзац"
    })
    .insertAfter("p.foo");
```

Полученный результат воспроизведет предыдущий (см. рис. 2.7).

### .wrap()

С помощью метода .wrap() разработчик может легко и быстро поместить существующий элемент в оболочку, образованную одним или несколькими новыми элементами.

В роли аргумента, принимаемого методом .wrap(), может выступать либо коллекция дескрипторов, образующих оболочку вокруг выбранных элементов, либо функция обратного вызова, генерирующая эти дескрипторы.

Для начала поместим все дескрипторы span в тестовом документе в дескрипторы strong, используя следующий код:

```
$("span").wrap("<strong />");
```

В результате выполнения этого кода текст внутри двух элементов span выделится полужирным шрифтом (рис. 2.8).

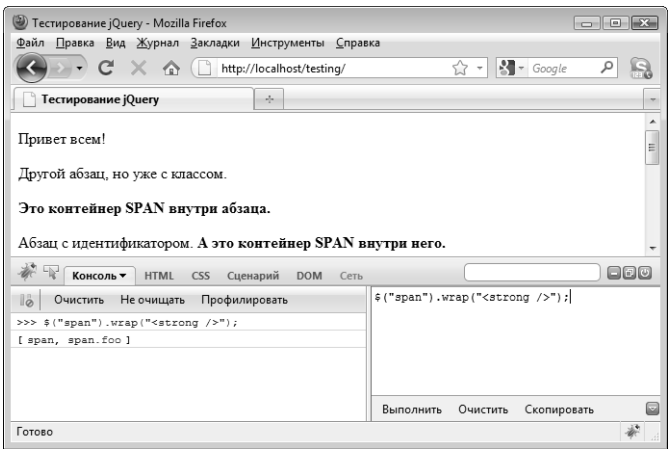



**Рис. 2.8.** Текст в контейнерах span, окруженных дескрипторами strong, отображается в браузере с использованием полужирного шрифта

Команда для помещения элемента внутрь оболочки имеет сравнительно свободный синтаксис, и результат, представленный на рис. 2.8, можно было бы получить с использованием любого из аргументов "", "" и "".

Кроме того, оболочкой могут служить сразу несколько дескрипторов, передаваемых методу .wrap() в виде набора вложенных дескрипторов.

```
$("span").wrap("<strong><em></em></strong>");
```

После выполнения этого кода текст внутри контейнеров `span` выделится в окне браузера полужирным курсивом (рис. 2.9).

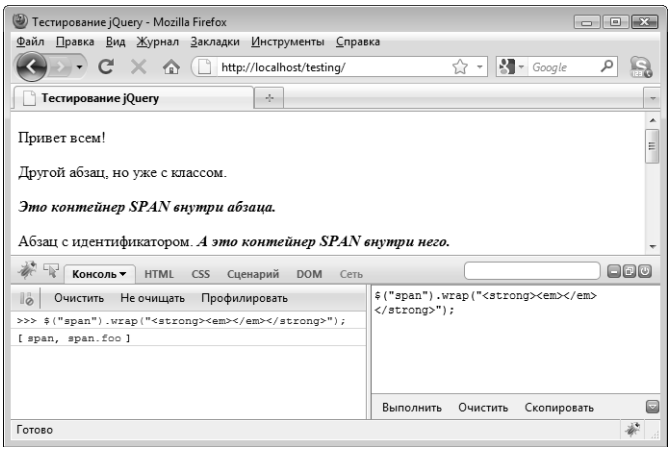



**Рис. 2.9.** Текст в контейнерах `span`, окруженных дескрипторами `strong` и `em`, отображается в браузере с использованием полужирного курсивного шрифта

Чтобы воспользоваться функцией обратного вызова для генерации требуемого дескриптора HTML, которым должен быть окружен элемент, вы должны возвратить из нее этот дескриптор. Например, чтобы окружить все элементы `span` с классом `foo` дескрипторами `strong`, а все остальные элементы `span` — дескрипторами `em`, используйте следующий код.

```
$("span").wrap(function(){
        return $(this).is(".foo") ? "<strong>" : "<em>";
    });
```

После выполнения этого кода текст в одном из контейнеров `span` отобразится в окне браузера курсивным шрифтом, а в другом (имеющем класс `foo`) — полужирным (рис. 2.10).

### *.unwrap()*

Метод `.unwrap()` выполняет обратную, по сравнению с методом `.wrap()`, задачу, удаляя дескрипторы, образующие оболочку вокруг заданного элемента. Он не принимает никаких аргументов, а просто находит ближайший родительский элемент и удаляет его.

Чтобы удалить в тестовом файле оболочки элементов `span`, образованные их родительскими элементами, выполните приведенный ниже код.

```
$("span").unwrap();
```

Этот код удалит родительские элементы (но оставит нетронутыми текстовые узлы), что приведет к изменению компоновки документа (рис. 2.11).

### *.wrapAll()*

Метод `.wrapAll()` используется для помещения набора элементов внутрь нового дескриптора. Вместо создания индивидуальных оболочек на основе нового дескрип-

тора вокруг каждого элемента текущего выбранного набора, он группирует все выбранные элементы и создает одну оболочку вокруг всей группы.

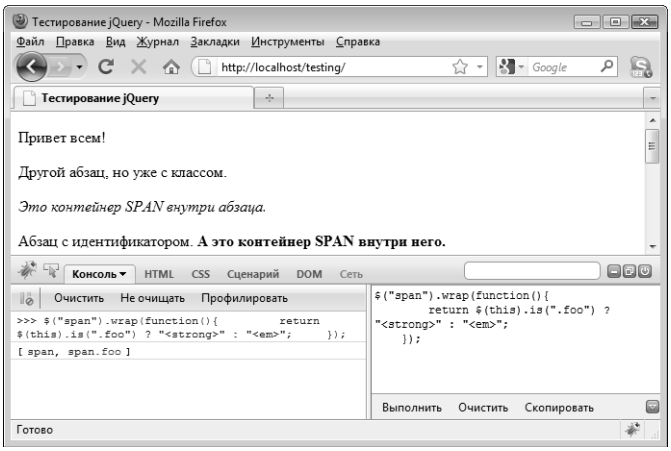



**Рис. 2.10.** Использование функции обратного вызова для заключения определенных дескрипторов в оболочку при выполнении определенных условий

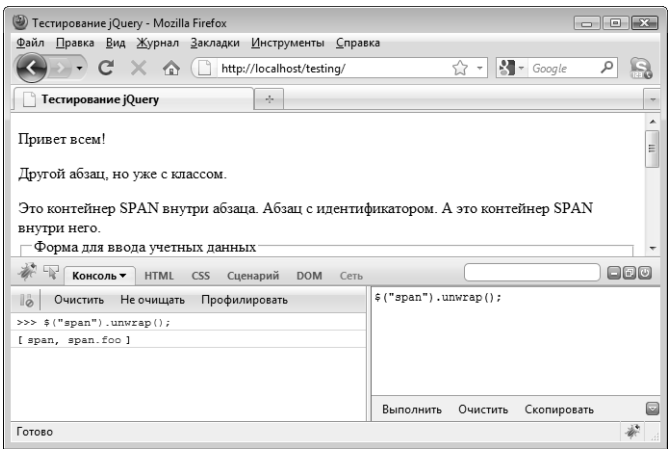



**Рис. 2.11.** Изменение компоновки документа после удаления ближайшей оболочки элементов `span`

Чтобы заключить все абзацы на странице в элемент `div` с желтым фоном, используйте приведенный ниже код.

```
var div = $("<div>", {
        "css":{"background-color":"yellow"}
    });
$("p").wrapAll(div);
```

После выполнения этого кода на странице появится новый элемент `div`, и все абзацы отобразятся внутри него на желтом фоне (рис. 2.12).

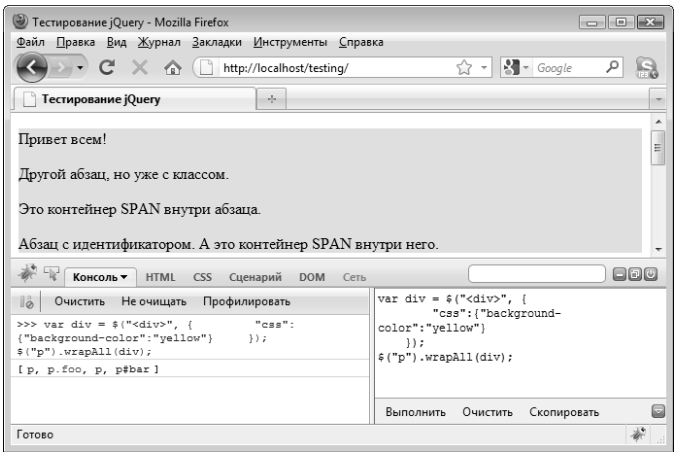



**Рис. 2.12.** Желтым фоном отмечены границы контейнера `div`, в который были успешно помещены все абзацы

Относительно метода `.wrapAll()` следует сделать одно важное замечание: группируя элементы, он перемещает их в DOM. Чтобы убедиться в этом, используйте метод `.wrapAll()` для заключения каждого абзаца документа в дескрипторы `strong`.

```
$("span").wrapAll("<strong />");
```

Обратите внимание на то, что после выполнения этой команды второй элемент `span` в документе был размещен в позицию вслед за первым, чтобы их можно было заключить в один дескриптор (рис. 2.13).

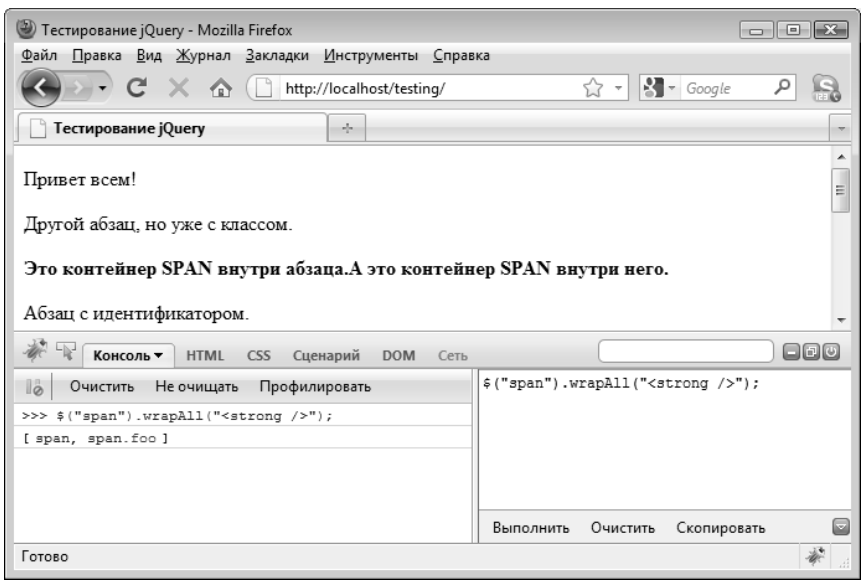



**Рис. 2.13.** Чтобы элементы `span` можно было заключить в один дескриптор, они размещаются один за другим

### .wrapInner()

Иногда необходимо заключить в оболочку не сам дескриптор, а только его содержимое. Неплохим примером этого может служить выделение полужирным шрифтом целого абзаца: простое окружение дескрипторами `strong` абзаца будет противоречить правилам HTML, и поэтому такое решение не подходит. К счастью, jQuery предоставляет метод `.wrapInner()`, который заключает в новый дескриптор все содержимое элемента.

Чтобы придать курсивное начертание тексту во всех абзацах на странице, используйте приведенный ниже код.

```
$("p").wrapInner("<em />");
```

После выполнения этого кода текст на странице будет отображаться курсивным шрифтом, а разметка будет вложена корректным образом (рис. 2.14).

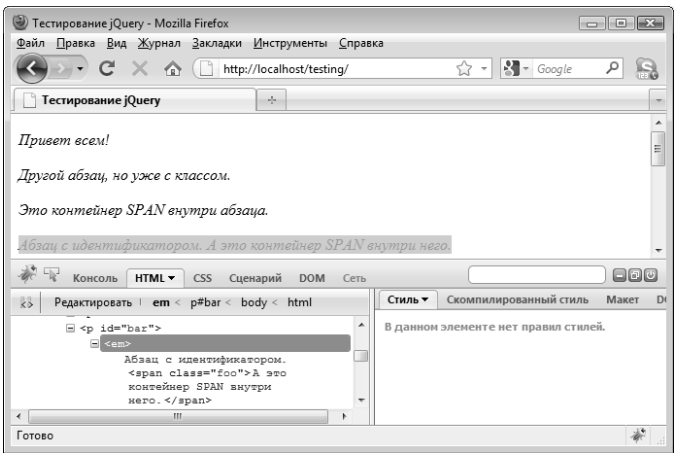



**Рис. 2.14.** Весь текст отображается курсивом, а все дескрипторы `em` находятся внутри дескрипторов `p`

### .remove() и .detach()

Методы `.remove()` и `.detach()` используются для полного удаления элемента из DOM. Оба метода удаляют выбранный элемент из DOM, но метод `.detach()` сохраняет данные jQuery, ассоциированные с удаляемым элементом, что делает этот метод идеальным для ситуаций, в которых предполагается последующее присоединение данного элемента к DOM в другой точке.

Как `.remove()`, так и `.detach()` принимают в качестве необязательного параметра селектор для нахождения элементов, подлежащих удалению. Удалите все абзацы с классом `foo` из нашего файла с примером, используя следующий код:

```
$("p").remove(".foo");
```

В результате выполнения этого кода абзац с классом `foo` исчезнет и больше не будет являться элементом DOM.

Чтобы продемонстрировать различие между методами `.remove()` и `.detach()`, нам придется забежать немного вперед и воспользоваться методом `.data()`, позволяющим присоединять информацию к элементу без введения дополнительных

дескрипторов или атрибутов (более подробно метод `.data()` рассматривается в следующем разделе).

Сначала мы ассоциируем некоторые данные с первым абзацем в DOM. Затем мы удалим элемент с добавленными данными из DOM с помощью метода `.detach()`, после чего заново присоединим удаленный элемент и попытаемся прочитать данные.

```
$("p:first").data("test","Это некоторые данные.");
var p = $("p:first").detach();
console.log("Сохраненные данные: "+p.data("test"));
```

**Примечание.** В этом примере для вывода служебной информации на консоль Firebug используются предусмотренные специально для Firebug объект `console` и его метод `.log()`. Их применение чрезвычайно полезно при отладке сценариев, но при передаче проекта в производственную среду они должны быть удалены из текста сценария во избежание появления ошибок JavaScript на компьютерах, на которых дополнение Firebug не установлено.

В процессе выполнения этого кода метод `.data()` присоединяет некоторую информацию к первому абзацу, удаляет его из DOM и сохраняет в переменной, после чего сценарий пытается вывести информацию, сохраненную в переменной с помощью метода `.data()`. В консоли отобразится следующий вывод.

```
>>>$("p:first").data("test","Это некоторые...данные:"+p.data("test"));
Сохраненные данные: Это некоторые данные.
```

А теперь выполните тот же тест, но на этот раз используйте метод `.remove()` вместо `.detach()`.

```
$("p:first").data("test","Это некоторые данные.");
var p = $("p:first").remove();
console.log("Сохраненные данные: "+p.data("test"));
```

В этом случае, как видно из консольного вывода, данные теряются вместе с удаляемым элементом.

```
>>>$("p:first").data("test","Это некоторые...данные:"+p.data("test"));
Сохраненные данные: null.
```

## Доступ к CSS и атрибутам и их изменение

Ранее при создании DOM-элементов мы могли определять различные атрибуты, такие как стили CSS, содержащийся внутри элемента текст и т.п. Для получения доступа к этой информации, относящейся к существующим элементам, в jQuery предусмотрен ряд встроенных методов.

### *.attr()*

Для работы с большинством атрибутов используется метод `.attr()`. Этот метод имеет двойное назначение. Во-первых, он позволяет прочитать заданный атрибут, используя имя требуемого атрибута в качестве первого и единственного в данном случае параметра. Во-вторых, с его помощью можно установить атрибут, передавая имя устанавливаемого атрибута в качестве первого параметра, а его значение — в качестве второго.

Сначала извлеките значение идентификатора (ID) последнего абзаца с помощью приведенного ниже кода.

```
$("p:eq(3)").attr("id");
```

При этом в консоли будет получен следующий результат.

```
>>>$("p:eq(3)").attr("id");
"bar"
```

Затем измените атрибут ID последнего абзаца на "bat", используя такой код:

```
$("#bar").attr("id", "bat");
```

После выполнения этого кода в консоли отобразится следующий вывод.

```
>>> $("#bar").attr("id", "bat");
[ p#bat ]
```

Если попытаться выбрать элементы с идентификатором bar, будет возвращен пустой результирующий набор.

```
>>> $("#bar");
[ ]
```

Однако теперь можно выбрать абзац с идентификатором bat.

```
>>> $("#bat");
[ p#bat ]
```

Кроме того, использование формата JSON позволяет задать сразу несколько атрибутов.

```
$("p:eq(3)").attr({
        "id":"baz",
        "title":"Очаровательный абзац, не правда ли"
    });
```

Открыв после выполнения этого кода панель HTML в Firebug, можно убедиться в том, что разметка абзаца изменилась.

```
<p id="baz" title="Очаровательный абзац, не правда ли?">
.removeAttr()
```

### *.removeAttr()*

Для удаления атрибута достаточно вызвать метод .removeAttr() для элемента, из которого вы хотите удалить атрибут, и передать имя атрибута в качестве параметра.

Активизируйте флажок в примере формы путем удаления атрибута disabled:

```
$(":checkbox").removeAttr("disabled");
```

После выполнения этого кода данный флажок можно будет устанавливать и снимать.

### *.css()*

Метод .css() работает аналогично методу .attr(), за исключением того, что он применяется к стилям. Чтобы изменить значение свойства стиля, передайте методу имя значения свойства в качестве единственного параметра; чтобы задать значение свойства, передайте методу как имя свойства, так и его новое значение. Как и в случае метода .attr(), использование формата JSON позволяет устанавливать сразу несколько значений.

Чтобы текст во всех элементах с классом `foo` отображался на желтом фоне красным цветом, используйте следующий код.

```
$(".foo").css({
        "color":"red", "background":"yellow"
    });
```

Этот код добавляет в выбираемые элементы новые свойства стилей (рис. 2.15).

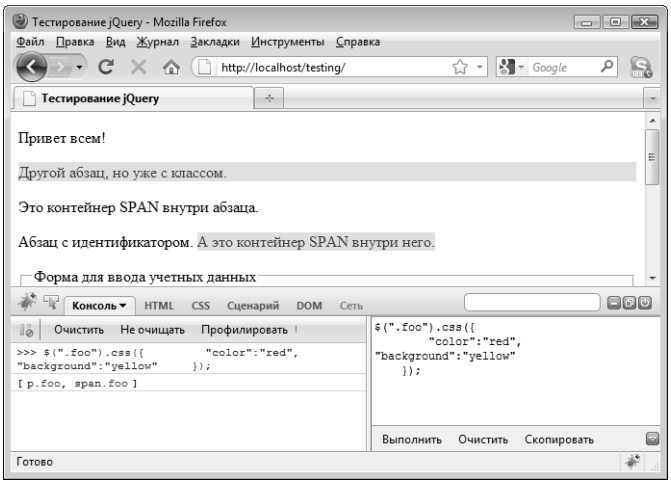



**Рис. 2.15.** Вид документа после добавления CSS-стиля в элементы с классом `foo`

Не перезагружая страницу, извлеките значение свойства `background` из элементов с классом `foo` с помощью такого кода:

```
$(".foo").css("background");
```

В консоли будет получен следующий результат.

```
>>> $(".foo").css("background");
"yellow none repeat scroll 0% 0%"
```

**Совет.** Возвращаемые значения представляют собой сокращенные обозначения свойств CSS[3]. Дополнительным преимуществом jQuery является возможность устанавливать свойства стилей с помощью сокращенных обозначений свойств CSS, чего не допускает базовый язык JavaScript.

### *.text() и .html()*

Для работы с содержимым элементов используются методы `.text()` и `.html()`. Различаются они между собой тем, что метод `.html()` предназначен для считывания и вставки HTML-дескрипторов, тогда как метод `.text()` предназначен лишь для считывания и записи текста.

При вызове элементом любого из этих методов без аргументов возвращается содержимое этого элемента. При передаче методу значения старое значение затирается и заменяется новым.

[3] `http://www.456bereastreet.com/archive/200502/efficient_css_with_shorthand_properties`.

Чтобы прочитать текст из абзаца с идентификатором `bar`, выполните в консоли следующий код:

```
$("#bar").text();
```

Этот код выбирает весь текст (включая пробельные символы), но игнорирует дескриптор `span`. Результирующий консольный вывод будет иметь следующий вид.

```
>>> $("#bar").text();
"Абзац с идентификатором.
        А это контейнер SPAN внутри него.
    "
```

Чтобы прочитать из абзаца все, что находится внутри него, включая и дескриптор `span`, используйте следующий код:

```
$("#bar").html();
```

Результирующий консольный вывод будет выглядеть так.

```
>>> $("#bar").html();
"Абзац с идентификатором.
        <span class="foo">А это контейнер SPAN внутри него.</span>
    "
```

А теперь изменим текст, передав его новое значение методу `.text()`.

```
$("#bar").text("Это новый текст.");
```

Предыдущее содержимое абзаца будет удалено, и вместо него появится новый текст. Обратите внимание на то, что одновременно был удален также дескриптор `span`: и `.text()`, и `.html()` удаляют *все содержимое* элемента.

Чтобы вставить HTML-разметку в абзац, вновь замените его содержимое с помощью следующего кода:

```
$("#bar").html("Это некоторый <strong>HTML</strong>текст.");
```

После выполнения этого кода в абзаце отобразится новый текст, в котором слово "HTML" будет выделено полужирным шрифтом (рис. 2.16).

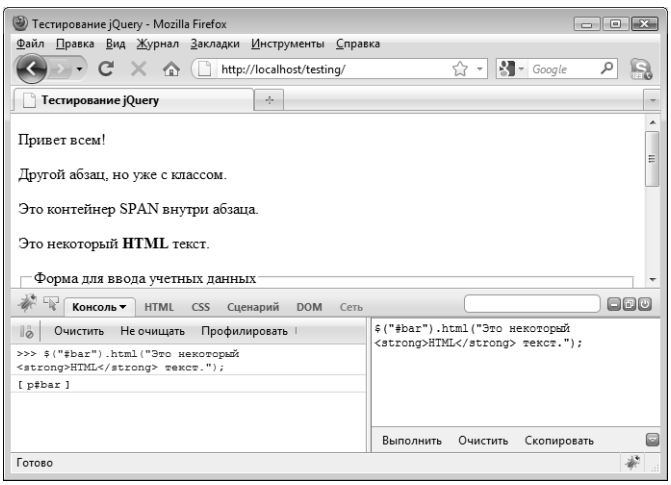



**Рис. 2.16.** Вид окна браузера после вставки текста и HTML-дескрипторов

### .val()

Доступ к содержимому элементов form и его изменение осуществляются посредством метода .val(). Этот метод возвращает значение элемента ввода или, если методу передается значение, устанавливает его.

Получите значение кнопки submit в примере формы с помощью следующего кода:

```
$(":submit").val();,
```

В консоли будет выведен следующий результат.

```
>>> $(":submit").val();
"Войти"
```

Затем обновите значение элемента ввода submit так, чтобы на кнопке отображался текст "Зарегистрироваться", воспользовавшись для этого следующим кодом:

```
$(":submit").val("Зарегистрироваться");
```

В результате выполнения этого кода на кнопке появится надпись "Зарегистрироваться".

### .data()

Ранее мы уже использовали метод .data() для сохранения информации при тестировании методов .remove() и .detach(). Именно для этого метод .data() и предназначен: сохранение информации об элементе в объекте jQuery безопасным и доступным способом.

Присвоим первым двум абзацам тестового документа имена, сохраним эту информацию с помощью метода .data(), а затем выведем ее на экране консоли.

```
$("p:first")
    .data("nickname", "Pookie")
    .next("p")
        .data("nickname", "Shnookums");
console.log("Меня зовут: "+$("p:first").data("nickname"));
console.log("Меня зовут: "+$("p:eq(1)").data("nickname"));
```

После выполнения этого сценария в консоли будет выведен следующий результат.

```
>>> $("p:first").data("nick...name:"+$("p:eq(1)").data("nickname"));
Меня зовут: Pookie
Меня зовут: Shnookums
```

Используя формат JSON, в элемент можно добавить *целую группу* данных.

```
$("p.foo").data({
    "nickname":"Wubby",
    "favorite":{
        "movie":"Pretty Woman",
        "music":"Sade", "color":"pink"
    }
});
console.log("Имя: "+$("p.foo").data("nickname"));
console.log("Любимый фильм:
"+$("p.foo").data("favorite").movie);
```

После выполнения этого кода в консоли будет получен следующий результат.

```
>>> $("p.foo").data({"nickname":"Wubby",....data("favorite").movie);
Меня зовут: Wubby
Любимый фильм: Pretty Woman
```

Это можно упростить, кешируя данные в переменной, например так.

```
$("p.foo").data({
    "nickname":"Wubby",
    "favorite":{
        "movie":"Pretty Woman",
        "music":"Sade",
        "color":"pink"
    }
});
var info = $("p.foo").data(); // кешируем объект данных
                              // в переменной
console.log("Меня зовут: "+info.nickname);
console.log("Любимый фильм: "+info.favorite.movie);
```

Результат аналогичен предыдущему, но этот код несколько лучше предыдущего и легче воспринимается.

### *.addClass(), .removeClass() и .toggleClass()*

Учитывая распространенность применения классов в современном веб-дизайне, для них были предусмотрены три удобных метода. Первые два из них, `.addClass()` и `.removeClass()`, предназначены соответственно для добавления и удаления атрибута класса.

```
$("p:first").addClass("bat");
console.log("Текст: "+$(".bat").text());
$("p:first").removeClass("bat");
console.log("Текст: "+$(".bat").text());
```

Выполнив этот фрагмент, вы получите в консоли следующий результат.

```
>>> $("p:first").addClass("bat"...le.log("Text: "+$(".bat").text());
Текст: Привет всем!
Текст:
```

Третий метод, `.toggleClass()`, принимает имя (или имена) класса, а затем либо добавляет класс, если он пока еще отсутствует в элементе, либо удаляет его, если класс уже существует.

Добавьте класс `baz` во второй абзац примера страницы и удалите из него класс `foo`, воспользовавшись следующим кодом:

```
$("p.foo").toggleClass("foo baz");
```

Выполнение этого кода изменяет абзац, удаляя из него старый класс и добавляя новый (рис. 2.17).

Чтобы вернуть обратно исходный класс `foo` и удалить класс `baz`, выберите абзац и вновь примените метод `.toggleClass()`.

```
$("p.baz").toggleClass("foo baz");
```

Выполнение этого кода возвращает абзац в прежнее состояние, в котором имеется только класс `foo`.

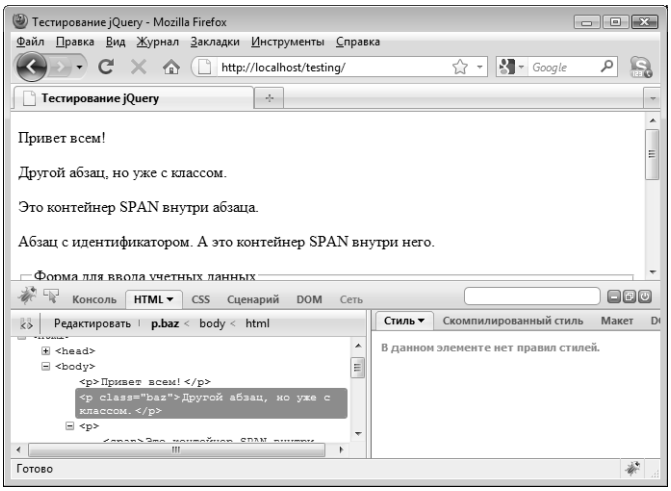



**Рис. 2.17.** Абзац с удаленным классом `foo` и добавленным классом `baz`

### *.hasClass()*

Метод `.hasClass()` работает аналогично методу `.is()` в том смысле, что также определяет, содержится ли данный класс в выбранном элементе, а затем возвращает значение `true` или `false`. Это делает его идеальным для применения в функциях обратного вызова.

Проверьте, содержится ли в первом абзаце класс `foo`, и выведите сообщение, текст которого зависит от результата такой проверки.

```
var msg = $("p:eq(1)").hasClass("foo") ? "Найден!" : "Нет!";
console.log("Класс? "+msg);
```

### *.height() и .width()*

Методы `.height()` и `.width()` удобны для определения соответственно высоты и ширины элемента. Оба они возвращают значения в виде целого числа без указания единиц измерения (если высота элемента составляет 68 пикселей, то метод `.height()` возвратит значение `68`). Этим он отличается от метода `.css()`, который возвращает также и единицу измерения.

Определите высоту и ширину формы с помощью следующего кода:

```
console.log("Высота формы: "+$("form").height()+"px");
```

Вы получите в консоли следующий результат.

```
>>> console.log("Высота формы:
"+$("form").height()+"px");
Высота формы: 249px
```

**Примечание.** Фактическое возвращенное значение высоты формы в вашем браузере может быть иным, в зависимости от типа установленной операционной системы.

Передача параметра методу `.height()` или `.width()` приводит к установке соответствующего значения. Установите для всех абзацев на странице высоту 100 пикселей и желтый фон с помощью такого кода:

```
$("p").height(100).css("background","yellow");
```

После выполнения этого кода высота всех абзацев изменится, а их фон станет желтым (рис. 2.18).

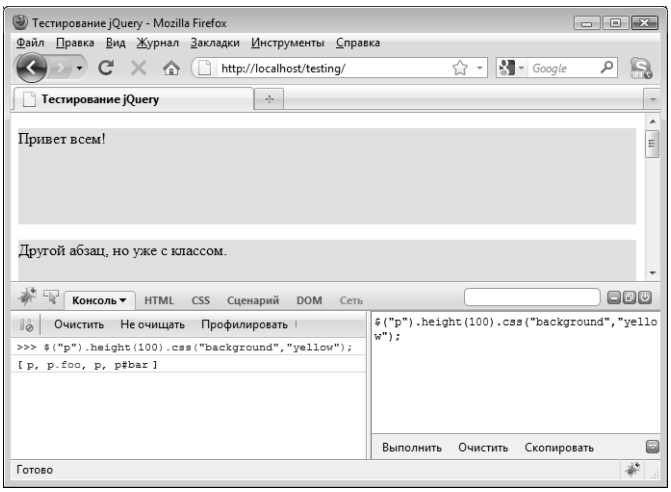



**Рис. 2.18.** Изменение высоты и фона для всех абзацев в документе

## *.innerHeight(), .innerWidth(), .outerHeight() и .outerWidth()*

Внутренняя высота или ширина элемента — это его высота или ширина без учета рамки и полей. Доступ к этой информации обеспечивают методы `.innerHeight()` и `.innerWidth()`.

Если вы хотите получить значения высоты или ширины элемента с учетом размеров окружающей его рамки, используйте методы `.outerHeight()` или `.outerWidth()`. Чтобы также включить размер полей, используйте методы `.outerHeight(true)` или `.outerWidth(true)`.

Добавьте в абзац с классом `foo` рамку и поля, а затем выведите значения его различных параметров высоты и ширины.

```
var el = $("p.foo");
el.css({
        "margin":"20px",
        "border":"2px solid black"
    });
console.log("Внутренняя ширина: "+el.innerWidth()+"px");
console.log("Внутренняя высота: "+el.innerHeight()+"px");
console.log("Внешняя ширина: "+el.outerWidth()+"px");
console.log("Внешняя высота: "+el.outerHeight()+"px");
console.log("Внешняя ширина с полями: "+el.outerWidth(true)+"px");
console.log("Внешняя высота с полями: "+el.outerHeight(true)+"px");
```

Результирующий вывод в консоли будет выглядеть следующим образом.

```
>>> var el = $("p.foo"); el.c...rgins: "+el.outerHeight(true)+"px");
Внутренняя ширина: 598px
Внутренняя высота: 20px
Внешняя ширина: 602px
Внешняя высота: 24px
Внешняя ширина с полями: 642px
Внешняя высота с полями: 64px
```

**Примечание.** Опять-таки, конкретные результаты зависят от используемой операционной системы.

## Воздействие на результирующие наборы

Для обработки наборов элементов нам потребуется ряд методов, позволяющих воздействовать на каждый элемент набора.

### *.map() и .each()*

Методы `.map()` и `.each()` позволяют разработчику предоставить собственную функцию обратного вызова, которая будет автоматически применена к каждому элементу выбранного набора. Эта функция должна иметь два аргумента: индекс текущего элемента и текущий DOM-элемент.

Различаются эти методы между собой тем, что возвращаемым значением метода `.map()` является новый объект, который содержит значения, возвращенные функцией обратного вызова, тогда как метод `.each()` возвращает исходный объект, элементы которого были изменены указанной функцией. Отсюда следует, что метод `.each()` можно включать в цепочки вызовов, а метод `.map()` — нельзя.

Чтобы найти все абзацы и элементы с классом `foo` и присоединить к каждому из них имя дескриптора и индекс элемента, используйте приведенный ниже код.

```
$("p,.foo").map(function(index, ele){
        $(this).append(" "+ele.tagName+" #"+index);
    });
```

Результатом выполнения этого кода будет добавление имени дескриптора элемента и индекса элемента в конец каждого из элементов выбранного набора (рис. 2.19).

Проделайте то же самое для метода `.each()`, заменив им в предыдущем коде метод `.map()`.

```
$("p,.foo").each(function(index, ele){
        $(this).append(" "+ele.tagName+" #"+index);
    });
```

Полученный результат совпадет с предыдущим.

Различия между этими двумя методами проявляются в тех случаях, когда требуется выполнить некоторую дополнительную обработку *после* применения метода `.map()` или `.each()`. Например, если вы хотите присоединить имя дескриптора и индекс к каждому абзацу и элементу с классом `foo`, как это уже было проиллюстрировано, а затем отобрать элементы `span` с классом `foo` и изменить для них цвет фона и текста, то вполне возможно, что вы попытались бы применить код наподобие следующего.

```
$("p,.foo").map(function(index, ele){
        $(this).append(" "+ele.tagName+" #"+index);
    })
```

```
    .find("span.foo")
        .css({
                "color":"red",
                "background":"yellow"
});
```

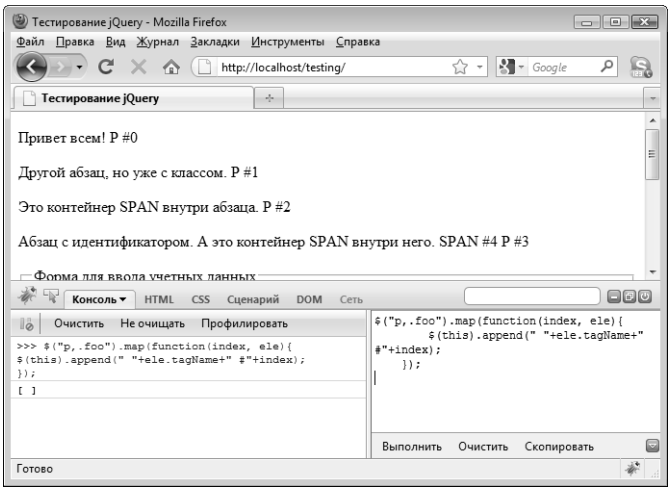



**Рис. 2.19.** Вид тестовой страницы после применения метода `map()` к функции обратного вызова для отображения имени и индекса каждого из найденных элементов

После применения этого кода имена дескрипторов и индексы присоединятся к элементам, но никакого изменения стиля элемента `span` не произойдет. Объясняется это тем, что объект, возвращенный методом `.map()`, уже не содержит ссылок на элементы.

Чтобы приведенный выше код заработал, как нам хочется, подставьте в него вместо вызова метода `.map()` вызов `.each()`.

```
$("p,.foo").each(function(index, ele){
        $(this).append(" "+ele.tagName+" #"+index);
    })
    .find("span.foo")
        .css({
                "color":"red",
                "background":"yellow"
            });
```

Теперь выполнение исправленного кода даст требуемый результат (рис. 2.20).

## Использование анимации и других эффектов

Одной из самых впечатляющих особенностей технологии jQuery является ее библиотека методов, позволяющая создавать анимацию и специальные эффекты, которые, хотя и могут быть воспроизведены обычными средствами JavaScript, несравненно легче реализуются с помощью jQuery. Традиционные методы JavaScript приводят к слишком сложным решениям, требующим определенного мастерства.

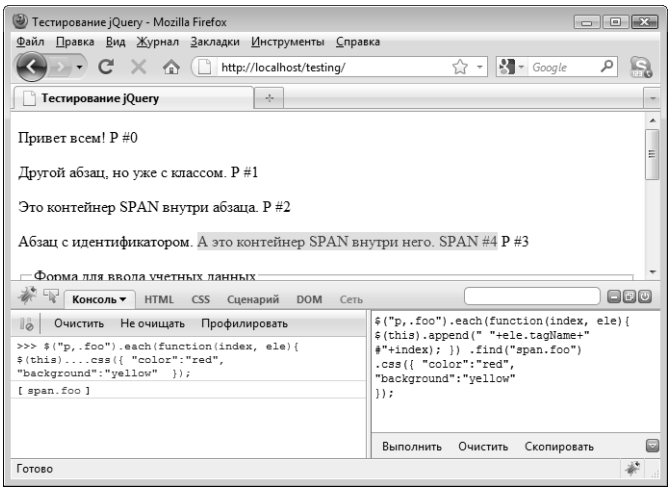



**Рис. 2.20.** Использование метода `each()` позволило получить требуемый результат

**Примечание.** Поскольку представить анимацию статическими изображениями весьма непросто, лишь браузер поможет вам увидеть приводимые ниже примеры в действии. Живую демонстрацию различных анимационных эффектов, обеспечиваемых различными методами, вы найдете на сайте jQuery API (`http://api.jquery.com`).

### *.show() и .hide()*

Простейшими функциями для создания анимационных эффектов являются функции `.show()` и `.hide()`. При запуске без параметра они просто добавляют свойство `display:none;` в атрибут стиля элемента или удаляют его.

Скройте абзац с идентификатором `bar`, используя следующий код:

```
$("#bar").hide();
```

Абзац исчезнет из окна браузера, но его по-прежнему можно будет увидеть в DOM с помощью средства просмотра элементов. Чтобы сделать элемент вновь видимым, вызовите метод `.show()`:

```
$("#bar").show();
```

В результате отображение элемента на экране восстановится.

Чтобы сокрытие и отображение элементов можно было использовать для создания анимационных эффектов, для обоих методов предусмотрена передача параметра, определяющего длительность перехода (в миллисекундах), а также функции обратного вызова, которая запускается по завершении анимации. Для демонстрации такого эффекта добавьте фон и рамку в абзац с идентификатором `bar`, а затем скройте его с длительностью перехода 2 секунды и вызовите функцию, которая выводит сообщение в окне консоли.

```
$("#bar")
    .css({
            "background":"yellow",
            "border":"1px solid black"
```

```
    })
.hide(2000,function(){
        console.log("Анимация завершена!");
    });
```

В процессе выполнения этого кода в элемент добавляются стилевые свойства CSS и запускается метод `.hide()`. Размеры элемента уменьшаются в горизонтальном и вертикальном направлениях, и он постепенно блекнет. Через две секунды элемент полностью исчезает, и функция обратного вызова выводит в окне консоли сообщение: `"Анимация завершена!"`.

**Примечание.** Функция обратного вызова запускается для каждого элемента анимируемого набора.

### *.fadeIn(), .fadeOut() и .fadeTo()*

Для создания (путем изменения параметра непрозрачности) эффекта плавного появления или исчезновения элемента на экране используются соответственно методы `.fadeIn()` и `.fadeOut()`. Эти методы изменяют значение параметра непрозрачности элементов либо от `0` до `1` (`fadeIn()`), либо от `1` до `0` (`fadeOut`). При исчезновении элемента к нему применяется свойство `display:none;`. При появлении элемента свойство `display:none;` удаляется из элемента, если оно присутствует.

Оба метода принимают в качестве аргументов необязательный параметр, определяющий длительность анимации (по умолчанию — 400 миллисекунд), и функцию обратного вызова, которая запускается по завершении анимации. Также предусмотрены два строковых параметра для задания двух фиксированных значений длительности, `"fast"` и `"slow"`, которые транслируются соответственно в значения `200` и `600` миллисекунд.

Чтобы создать эффект исчезновения формы, вывести сообщение, а затем создать эффект появления формы и вывести другое сообщение, используйте следующий код.

```
$("form")
    .fadeOut(1000, function(){
            console.log("Форма исчезла!");
        })
    .fadeIn(1000, function(){
            console.log("Форма появилась!");
        });
```

Третий метод из этого семейства, `.fadeTo()`, позволяет указать минимальный уровень, до которого должно снизиться значение параметра непрозрачности. Этому методу требуются два параметра: длительность перехода и предельное значение параметра непрозрачности (число между `0` и `1`). Кроме того, в качестве необязательного третьего параметра может передаваться функция обратного вызова.

Создайте эффект плавного уменьшения непрозрачности формы до 50 % и выведите сообщение, используя следующий код.

```
$("form")
    .fadeTo(1000, 0.5, function(){
            console.log("Непрозрачность снижена до 50%!");
        });
```

### *.slideUp(), .slideDown() и .slideToggle()*

Метод `.slideUp()` позволяет плавно уменьшить высоту элемента до `0`. Как только высота элемента достигает нулевого значения, ему присваивается свойство

display:none, исключающее влияние элемента на компоновку страницы. Метод .slideDown() выполняет обратное действие: он удаляет свойство display:none и плавно увеличивает высоту элемента от 0 до ее исходного значения.

Аналогично методам .fadeIn() и .fadeOut(), данные методы принимают два необязательных параметра: длительность анимации и функцию обратного вызова. В качестве демонстрационного примера сверните абзац с классом foo по высоте, выведите сообщение в окне консоли, а затем разверните абзац до его первоначального размера и вновь выведите соответствующее сообщение.

```
$("p.foo")
    .slideUp(1000, function(){
            console.log("Скрыт!");
        })
    .slideDown(1000, function(){
            console.log("Отображен!");
        });
```

Метод .slideToggle() делает то же самое, что и методы .slideUp() и .slideDown(), достаточно "интеллектуален" и способен самостоятельно определить, какое действие необходимо выполнить, в зависимости от того, скрыт элемент или отображен.

Для проверки того, как работает этот метод, примените его к абзацу с классом foo.

```
$("p.foo")
    .slideToggle("slow", function(){
            console.log("Toggled!");
        });
```

Выполнив этот код последовательно несколько раз, вы увидите, как абзац попеременно то свертывается по высоте, то развертывается.

### .animate()

Методы анимации, которые мы до сих пор обсуждали, являются рационализированными способами вызова метода .animate() для разных частных случаев. Сам же метод .animate() способен анимировать большинство CSS-свойств элемента и может принимать в качестве одного из параметров так называемую функцию *изинга* (easing), определяющую способ выполнения анимационного перехода. По умолчанию поддерживаются строковые параметры изинга "linear" и "swing", но для jQuery доступны и другие функции изинга в виде легко подключаемых дополнений (более подробно о дополнениях говорится далее).

Метод .animate() может принимать несколько аргументов, для передачи которых существует два способа. Первый способ предполагает передачу набора анимируемых CSS-свойств в формате JSON в качестве первого аргумента, а также трех дополнительных аргументов, каждый из которых не является обязательным: длительности анимации (в миллисекундах), формулы, описывающей функцию изинга, и функции обратного вызова. Второй способ предполагает передачу набора CSS-свойств в формате JSON в качестве первого аргумента и набора параметров в формате JSON в качестве второго аргумента.

Используем первый способ вызова метода animane(), чтобы установить фон и стиль рамки абзаца с идентификатором bar, а затем анимировать ширину и высоту этого абзаца на протяжении 5 секунд с использованием изинга типа "swing" с последующим выводом сообщения после завершения анимации с помощью такого кода.

```
$("#bar")
    .css({
            "background":"yellow",
            "border":"1px solid black"
        })
    .animate({
            "width":"500px",
            "height":"100px"
        },
        5000,
        "swing",
        function(){
                console.log("Анимация завершена!");
            });
```

После выполнения этого кода абзац отобразится на желтом фоне в черной рамке, а его размеры изменятся в соответствии с переданными параметрами (рис. 2.21).

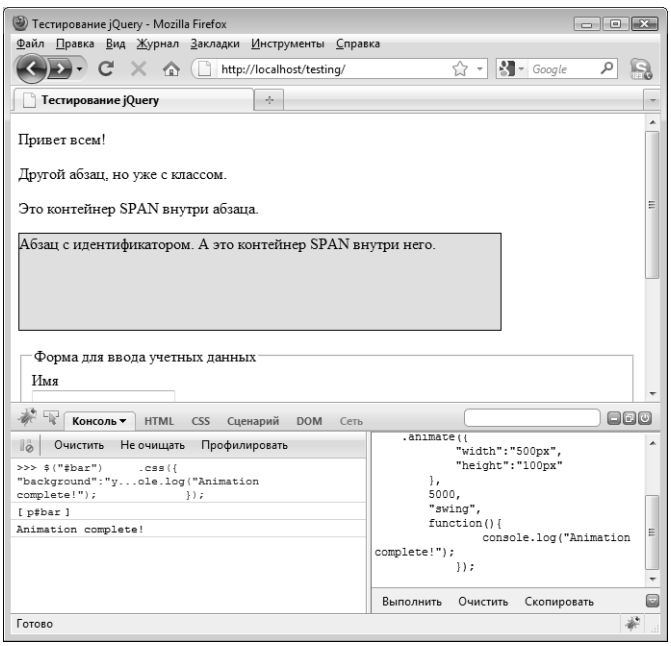



**Рис. 2.21.** Вид абзаца по окончании процесса анимации его ширины и высоты

При использовании второго способа вызова метода `animate()` соответствующий код выглядел бы следующим образом.

```
$("#bar")
    .css({
            "background":"yellow",
            "border":"1px solid black"
        })
    .animate({
            "width":"500px",
            "height":"100px"
```

```
        },
        {
            "duration":5000,
            "easing":"swing",
            "complete":function(){
                console.log("Анимация завершена!");
            }
        });
```

Результат выполнения этого кода будет таким же. Второй формат передачи параметров методу `animate()` предоставляет дополнительные возможности. Для выполнения тех же действий, но с использованием всех доступных возможностей, можно было бы применить код наподобие следующего.

```
$("#bar")
    .css({
            "background":"yellow",
            "border":"1px solid black"
        })
    .animate({
            "width":"500px",
            "height":"100px"
        },
        {
            "duration":5000,
            "easing":"swing",
            "complete":function(){
                    console.log("Анимация завершена!");
                },
            "step":function(){
                    console.log("Этап завершен!");
                },
            "queue":true,
            "specialEasing":{
                    "width":"linear"
                }
        });
```

Параметр `step` позволяет разработчику создать функцию, которая будет запускаться по завершении каждого этапа анимации. Это происходит при любом изменении свойства, поэтому в консоли отобразится довольно много сообщений `"Этап завершен!"`.

Параметр `queue` указывает, должна ли анимация включаться в текущую очередь, т.е. позволяет задать порядок вызова анимаций. В случае вызова нескольких анимаций и помещения их в очередь вторая анимация начнется лишь после того, как закончится первая, третья — лишь после того, как закончится вторая, и т.д.

Параметр `specialEasing` позволяет присоединять к каждому из анимируемых CSS-свойств различные стили изинга.

**Примечание.** Параметр `specialEasing`, детище Джеймса Падолси (James Padolsey), впервые появился в jQuery 1.4. Великолепный пример его применения можно найти по такому адресу: `http://james.padolsey.com/demos/jquery/easing/easing-jq14.html`

### .delay()

Метод .delay(), впервые появившийся в jQuery 1.4, позволяет задать длительность паузы в выполнении сценария, выраженную в миллисекундах. Это позволяет выполнить одну анимацию и организовать некоторую задержку, прежде чем начать выполнение другой анимации.

Сверните абзац с идентификатором bar по высоте, выждите 3 секунды, а затем разверните абзац до прежнего размера, используя следующий код.

```
$("#bar")
    .css({
            "background":"yellow",
            "border":"1px solid black"
        })
    .slideUp(1000, function(){
            console.log("Анимация завершена!");
        })
    .delay(3000)
    .slideDown(1000, function(){
            console.log("Анимация завершена!");
        });
```

### .stop()

Для остановки анимации используется метод .stop(). Этот метод принимает два булевых аргумента: один определяет, следует ли очистить очередь, а второй — следует ли немедленно перейти к концу анимации. Для обоих аргументов значением по умолчанию является false.

Запустите анимацию, прекратите анимацию, очистите очередь и организуйте немедленный переход к концу анимации после выполнения 200 этапов, используя следующий код.

```
var count = 0; // Отслеживаем текущее значение счетчика
$("#bar")
    .css({
            "background":"yellow",
            "border":"1px solid black"
        })
    .animate({
                "width":"500px"
            },
        {
                "duration":6000,
                "step":function(){
                        if(count++==200)
                        {
                            $(this).stop(true, true);
                        }
                    }
            });
```

# Обработка событий

Во многих сценариях желательно иметь возможность выполнять заданные действия при наступлении определенных событий или совершении определенных действий в браузере. В jQuery имеется встроенная поддержка событий браузера, что и является предметом обсуждения в следующих разделах.

## *События браузера*

События браузера происходят тогда, когда в самом браузере возникают некоторые изменения или ошибки.

### `.error()`

Это событие запускается при возникновении ошибок в браузере. Одной из обычных ситуаций, влекущих за собой возникновение ошибки в браузере, является наличие дескриптора, пытающегося загрузить несуществующее изображение. Метод `.error()` позволяет связать с событием некий обработчик (т.е. функцию, которая запускается в ответ на наступление события).

Создайте дескриптор, пытающийся отобразить несуществующее изображение, и подключите к событию ошибки обработчик, который просто выводит сообщение на экране консоли.

```
$("<img />", {
        "src":"not/an/image.png",
        "alt":"Такого изображения не существует"
    })
    .error(function(){
            console.log("Не удается загрузить изображение!");
        })
    .appendTo("body");
```

После выполнения этого кода на экране консоли отобразится следующая информация.

```
>>> $("<img />", { "src":"not/an/image.png", ...изображение!");})
.appendTo("body");
[ img image.png ]
Не удается загрузить изображение!
```

### `.scroll()`

При прокрутке документа генерируется событие `scroll`. Для связывания обработчика с этим событием используется метод `.scroll()`.

```
$(window)
    .scroll(function(){
            console.log("Осуществлялась прокрутка окна!");
        });
```

Если после выполнения этого кода вы будете прокручивать окно браузера, то в окне консоли каждый раз будет выводиться соответствующее сообщение об этом.

Вызов метода `.scroll()` без параметров запускает событие `scroll`. Предварительно связав с окном обработчик событий, как показано в предыдущем примере, запустите это событие с помощью такого кода:

```
$(window).scroll();
```

В результате выполнения этого кода на экране консоли отобразится сообщение о прокрутке окна.

### Обработка событий загрузки документа

В JavaScript часто требуется дождаться полной загрузки документа, прежде чем дать возможность выполниться какому-либо сценарию. Кроме того, иногда, когда пользователь покидает страницу, желательно запускать функцию, которая позволяет удостовериться, что пользователь действительно намеревается выполнить такой переход.

#### .ready()

Почти в каждом сценарии jQuery для исключения возможности преждевременного, а значит, и не соответствующего исходному плану выполнения сценария используется метод `.ready()`. Этот метод гарантирует запуск переданного ему обработчика лишь после того, как DOM-модель будет полностью подготовлена к последующим манипуляциям.

Обычная практика состоит в использовании всего сценария в качестве функции обратного вызова, запускаемой обработчиком `.ready()`.

```
$(document).ready(function(){
        // Вся функциональность jQuery находится здесь});
```

Кроме того, метод `.ready()` может принимать параметр, указывающий псевдоним для функции `jQuery`. Это позволяет создавать защищенные от сбоев сценарии jQuery, которые будут нормально работать, даже если псевдоним `$` передается в другую библиотеку с использованием вызова `jQuery.noConflict()` (что позволяет без труда подключать к одному проекту сразу несколько библиотек JavaScript, в которых используется псевдоним `$`). Можно гарантировать правильную работу псевдонима `$`, используя следующий код.

```
jQuery(document).ready(function($){
        // Вся функциональность jQuery находится здесь
        $("p").fadeOut();
    });
```

С технической точки зрения здесь может передаваться любой псевдоним.

```
jQuery(document).ready(function(xTest){
        xTest("#bar").click(function(){console.log("Щелчок!");});
    });
```

Как и следовало ожидать, этот код также работает безукоризненно. Существует не так уж много случаев, требующих принятия подобных мер предосторожности, однако этот пример иллюстрирует, как работают псевдонимы с методом `.ready()`. Наконец, сама функция `jQuery` может быть использована в качестве псевдонима для метода `.ready()`.

```
jQuery(function($){
        // действия, предпринимаемые после построения DOM
    });
```

#### .unload()

Всякий раз, когда пользователь покидает страницу путем щелчка на ссылке, перезагрузки страницы, с помощью кнопок Вперед (Forward) или Назад (Back) или полного закрытия окна браузера, запускается событие `unload`. Однако в разных

браузерах обработка события `unload` осуществляется по-разному. *Поэтому, прежде чем использовать это событие в производственных сценариях, следует протестировать код в различных браузерах.* Создайте ссылку на сайт Google и подключите к событию `unload` вывод окна сообщения с помощью следующего кода.

```
$("<a>", {
            "href":"http://google.com",
            "text":"Перейти на Google!"
        })
    .appendTo("#bar");
$(window).unload(function(){
        alert("Пока! Можете немного 'погуглить'! ");
    });
```

Выполните этот код и щелкните на новой ссылке. На экране отобразится окно сообщения, и вы будете перенаправлены на страницу Google.

## Подключение событий

Существует множество событий браузера, запускаемых пользователем, и jQuery предоставляет ряд методов, позволяющих легко их обрабатывать.

Доступны события `blur`, `focus`, `focusin`, `focusout`, `load`, `resize`, `scroll`, `unload`, `click`, `dblclick`, `mousedown`, `mouseup`, `mousemove`, `mouseover`, `mouseout`, `mouseenter`, `mouseleave`, `change`, `select`, `submit`, `keydown`, `keypress`, `keyup` и `error`.

### `.bind()` и `.unbind()`

Для связывания обработчика событий с элементом используется метод `.bind()`. Этот метод принимает событие в качестве своего первого аргумента и функцию обработчика — в качестве второго.

В качестве первого аргумента можно указать список событий, разделенных пробелами, что позволяет связать с элементом сразу несколько событий. Кроме того, для связывания различных событий с различными обработчиками методу `.bind()` можно передавать объект в формате JSON. Свяжите вывод сообщения в консоли с событием `click`, используя следующий код.

```
$("p")
    .bind("click", function(){
        console.log("Выполнен щелчок!");
    });
```

Щелчок на любом абзаце после выполнения этого кода приведет к выводу сообщения в окне консоли. Чтобы связать обработчик как с событием `click`, так и с событием `mouseover`, выполните следующий код.

```
$("p")

    .bind("click mouseover",function(){
            console.log("Произошло событие!");
        });
```

Теперь и щелчок на абзаце, и наведение на него указателя мыши вызовут появление сообщения о событии в окне консоли.

Чтобы обеспечить возможность передачи данных обработчику, был предусмотрен дополнительный параметр. Данный параметр имеет вид объекта в формате JSON, содержащего переменные, которые должны использоваться в функции. Эти

переменные связаны с объектом события, так что их значения остаются неизменными в пределах данного обработчика.

Установите обработчик щелчка для двух абзацев тестового документа так, чтобы при выполнении щелчков на них выводились разные сообщения, воспользовавшись для этого следующим кодом.

```
// Создать значение для переменной notice
var notice = "Я хранюсь в переменной!";
$("p.foo").bind("click", { n:notice }, function(event){
        console.log(event.data.n);
    });

// Изменить значение переменной "notice"
var notice = "Я также хранюсь в переменной!";
$("#bar").bind("click", { n:notice }, function(event){
        console.log(event.data.n);
    });
```

Чтобы привязать к событиям `click` и `mouseover` разные обработчики, используйте следующий код.

```
$("p")
    .bind({
            "click":function(){
                    console.log("Щелчок!");
                },
            "mouseover":function(){
                    console.log("Наведение указателя мыши!");
                }
        });
```

После выполнения этого кода в консоли будут выводиться разные сообщения для разных событий при их наступлении.

Для удаления любого события достаточно вызвать метод `.unbind()`. Будучи вызванным без параметров, он удаляет из элемента все события. Чтобы уточнить, какое именно событие подлежит удалению, можно указать его имя в качестве первого параметра. Чтобы дополнительно уточнить, какую именно функцию следует удалить из события, можно указать ее имя в качестве второго параметра.

Чтобы отключить все события от абзацев в нашем примере, используйте следующий код:

```
$("p").unbind();
```

Для удаления лишь обработчика щелчка используйте следующий код:

```
$("p").unbind("click");
```

Если же с элементом была связана какая-либо специальная функция, ее можно удалить следующим образом.

```
var func1 = function(){
            console.log("Запуск события!");
        },
    func2 = function(){
            console.log("Другой обработчик!");
        };
$("#bar")
    .bind("click", func1)
```

```
    .bind("click", func2)
    .trigger("click") // запуск только одного события
    .unbind("click", func1);
```

Приведенный выше код создает две функции (сохраняемые в двух переменных `func1` и `func2`), связывает их с событием `click` для абзаца с идентификатором `bar`, однократно запускает событие (о функции `.trigger()` говорится далее) и отключает функцию, хранящуюся в переменной `func1`.

После выполнения данного кода щелчок на абзаце будет запускать лишь функцию, хранящуюся в переменной `func2`.

### `.live()` и `.die()`

Аналогично методам `.bind()` и `.unbind()`, методы `.live()` и `.die()` предназначены соответственно для подключения обработчиков событий к элементу или их удаления. Основным отличием является то, что метод `.live()` осуществляет подключение обработчиков событий и свойств JavaScript не только к существующим элементам, но и к любым новым элементам, динамически добавленным в DOM, которые соответствуют селектору.

Например, добавьте обработчик события `click` для всех элементов `a` с помощью такого кода.

```
$("a")
    .live("click", function(){
            console.log("Щелчок на ссылке!");
            return false; // предотвратить запуск события при щелчке на ссылке
        });
```

Конечно же, в настоящий момент на странице из нашего примера отсутствуют какие-либо ссылки. Не перезагружая страницу, добавьте дескриптор `a` в абзац с идентификатором `bar` с помощью следующего кода.

```
$("<a>", {
            "href":"http://google.com",
            "text":"Перейти на Google!"
        })
    .appendTo("#bar");
```

На странице отобразится новая ссылка, и хотя привязка события была осуществлена еще до того, как в DOM имелся хотя бы один дескриптор `a`, щелчок на ссылке вызовет вывод сообщения в окне консоли, и перехода по ссылке не произойдет.

Выполнение предыдущего действия с использованием метода `.bind()` не даст желаемого результата. Кроме того, обработчик события `click`, связанный с методом `.live()`, нельзя удалить с помощью метода `.unbind()`; для удаления события следует привлечь метод `.die()`. Метод `.die()` используется точно так же, как и метод `.unbind()`.

### `.one()`

Назначение и способ использования методов `.one()` и `.bind()` совпадают, за исключением того, что метод `one()` отключает обработчик событий лишь после того, как событие успеет произойти один раз. Добавьте в абзац с идентификатором `bar` новый обработчик событий, который запустится только один раз, используя для этого следующий код.

```
$("#bar").one("click", function(){
       console.log("Этот обработчик запустится только один раз.");
    });
```

После выполнения этого кода щелчок на абзаце с идентификатором `bar` будет сопровождаться выводом сообщения в окне консоли только один раз, тогда как все последующие щелчки не будут иметь никакого эффекта.

### .toggle()

Метод `.toggle()` позволяет связать с событием `click` две и более функций, которые будут поочередно вызываться при последовательном выполнении щелчков. Этот метод может использоваться для попеременного изменения видимости элементов, например путем переключения между методами `.show()` и `.hide()`, аналогично тому, как последовательные вызовы метода `.slideToggle()` обеспечивают поочередный вызов методов `.slideUp()` и `.slideDown()`.

Прежде всего, свяжите с событием щелчка на абзаце с идентификатором `bar` три различные функции для вывода сообщений, используя следующий код.

```
$("#bar")
    .toggle(function(){
            console.log("Функция 1");
            },
            function(){
            console.log("Функция 2");
            },
            function(){
            console.log("Функция 3");
            });
```

После выполнения этого кода последовательные щелчки на абзаце с идентификатором `bar` будут сопровождаться поочередным выводом трех различных сообщений в окне консоли. Далее организуйте переключение видимости абзаца с идентификатором `bar` с помощью следующего кода:

```
$("#bar").toggle();
```

Запуск этой команды скроет абзац. Повторный ее запуск восстановит видимость абзаца. Если при вызове метода указать длительность анимации в качестве первого аргумента, то сокрытие и восстановление видимости элемента будут происходить в режиме анимации.

```
$("#bar").toggle(2000);
```

Наконец, передача методу булевого флага позволяет указать, что именно следует сделать: отобразить или скрыть элементы.

```
$("#bar").toggle(true); // будут отображены все элементы
$("#bar").toggle(false); // будут скрыты все элементы
```

### .trigger()

Для запуска событий используется метод `.trigger()`. Этот метод принимает событие, которое следует запустить, и необязательный массив аргументов, подлежащих передаче обработчику. Подключите обработчик к абзацу с идентификатором `bar` и запустите его, используя следующий код.

```
$("#bar")
    .bind("click", function(){
            console.log("Щелчок!");
        })
    .trigger("click");
```

Чтобы передать дополнительные данные, измените код, как показано ниже.

```
// создаем переменную
var note = "Запуск события!";
$("#bar")
    .bind("click", function(event, msg){ // разрешен 2-й
                                         // аргумент
            // Если переменная "msg" не передается,
            // выводится сообщение, заданное по умолчанию
            var log = msg || "Щелчок!";
            console.log(log);
        })
    .trigger("click", [ note ]); // передаваемый массив указан в
                                 // квадратных скобках
```

Этот код выведет в окне консоли сообщение, хранящееся в переменной `note`.

### Прямые методы для работы с событиями

Для каждого события предусмотрен прямой метод (shortcut), который принимает функцию обработчика в качестве аргумента. Если такой метод вызывается без передачи аргумента, то он вызывает метод `.trigger()` для своего типа события.

К числу доступных прямых методов относятся `.blur()`, `.focus()`, `.focusin()`, `.focusout()`, `.load()`, `.resize()`, `.scroll()`, `.unload()`, `.click()`, `.dblclick()`, `.mousedown()`, `.mouseup()`, `.mousemove()`, `.mouseover()`, `.mouseout()`, `.mouseenter()`, `.mouseleave()`, `.change()`, `.select()`, `.submit()`, `.keydown()`, `.keypress()`, `.keyup()` и `.error()`.

В качестве примера ниже показан код, связывающий обработчик с событием `click` и запускающий событие.

```
$("#bar").click(function(){ console.log("Щелчок!"); }).click();
```

## Использование элементов управления AJAX

Последний из наборов методов jQuery, с которым мы намерены ознакомиться, является, пожалуй, наиболее полезным, и, скорее всего, именно он сыграл значительную роль в столь широком распространении jQuery. Методы, предоставляющие функциональные возможности AJAX[4], оказались чрезвычайно полезными и весьма простыми в использовании, что особенно важно для тех, кто ранее уже создавал сценарии AJAX в рамках JavaScript.

**Примечание.** Если вы хотите больше узнать об AJAX, обратитесь к статье в Википедии по следующему адресу: `http://ru.wikipedia.org/wiki/Ajax`

Для работы с примерами в этом разделе вам понадобится внешний файл, доступ к которому будет осуществляться с помощью элементов управления AJAX. Создайте в папке `testing` новый файл `ajax.php`. Введите в него следующий код.

```
<?php
echo '<p class="ajax">Этот абзац загружен с помощью AJAX.</p>',
'<pre>GET variables: ', print_r($_GET, TRUE), '</pre>',
'<pre>POST variables: ', print_r($_POST, TRUE), '</pre>';
?>
```

[4] `http://en.wikipedia.org/wiki/Ajax_(programming)`.

Мы будем обращаться к этому файлу, привлекая различные методы AJAX, доступные в jQuery. Он будет использоваться нами в демонстрационных целях для отображения передаваемых сценарию данных.

### *$.ajax()*

**Низкоуровневой** функцией для передачи запросов AJAX является функция `$.ajax()`. Обратите внимание на то, что эта функция вызывается без селектора, поскольку она не применяется к объекту jQuery. Все операции AJAX выполняются глобальными функциями, не зависящими от DOM.

Функция `$.ajax()` принимает один аргумент — объект, содержащий параметры вызова AJAX. Если этот аргумент опущен, функция загрузит текущую страницу, не выполняя над результатом никаких дополнительных действий.

Общее количество доступных для функции `$.ajax()` параметров довольно велико, однако не все они рассматриваются или используются в данной книге. Полный список параметров можно найти по адресу `http://api.jquery.com/jQuery.ajax`. Список наиболее употребительных параметров приводится ниже.

- **`data`.** Данные, которые должны быть переданы удаленному сценарию, записанные либо в виде строки запроса (`key1=val1&key2=val2`), либо в формате JSON (`{"key1":"val1","key2":"val2"}`).
- **`dataFilter(data, type)`.** Функция обратного вызова, которая позволяет выполнять предварительную фильтрацию данных и великолепно подходит для очистки данных, поступающих от удаленного сценария.
- **`dataType`.** Описание типа данных, ожидаемых из запроса. В случае его отсутствия jQuery в состоянии делать разумные предположения относительно этого. Доступными типами данных являются `"xml"`, `"html"`, `"script"`, `"json"`, `"jsonp"` и `"text"`.
- **`error(XMLHttpRequest, textStatus, errorThrown)`.** Функция обратного вызова, которая должна выполняться в случае возникновения ошибок запроса. Аргументами функции служат объект `XMLHttpRequest`, строка состояния запроса и код ошибки.
- **`success(data, textStatus, XMLHttpRequest)`.** Функция обратного вызова, которая должна выполняться в случае успешного завершения запроса. Аргументами функции служат данные, возвращаемые удаленным сценарием, строка состояния запроса и объект `XMLHttpRequest`.
- **`type`.** Тип отправляемого запроса. По умолчанию таковым является `GET`, но доступен и тип `POST`. Также допускается использование типов `PUT` и `DELETE`, но их правильная работа со всеми браузерами не гарантируется.
- **`url`.** Адрес URL, по которому отправляется запрос.

Для отправки простейшего POST-запроса тестовому сценарию и загрузки результатов в абзац с идентификатором `bar` вполне подойдет следующий код.

```
$.ajax({
        "type":"POST",
        "url":"ajax.php",
        "data":"var1=val1&var2=val2",
        "success":function(data){
                $("#bar")
                    .css("background","yellow")
```

```
                .html(data);
        }
});
```

В результате выполнения этого кода содержимое абзаца будет заменено загруженной информацией (рис. 2.22).

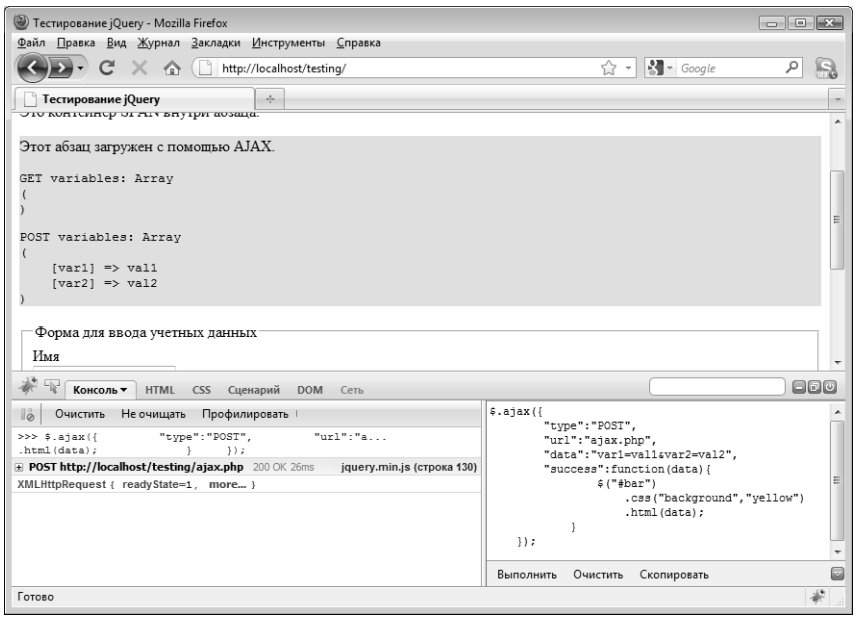



**Рис. 2.22.** Информация AJAX, загруженная из файла `ajax.php`

## *$.ajaxSetup()*

Значения параметров по умолчанию для вызовов AJAX устанавливаются с помощью функции `$.ajaxSetup()`. Например, для указания того, что по умолчанию все запросы AJAX должны направляться файлу `ajax.php` посредством процедуры POST, а затем загружаться в абзац с идентификатором `bar`, можно использовать следующий код.

```
$.ajaxSetup({
        "type":"POST",
        "url":"ajax.php",
        "success":function(data){
                $("#bar")
                    .css("background","yellow")
                    .html(data);
        }
});
```

Теперь будет очень легко создавать новые запросы AJAX, просто передавая новые данные.

```
$.ajax({
        "data":{
                "newvar1":"value1",
```

```
            "newvar2":"value2"
        }
});
```

Это приведет к замене содержимого абзаца новым содержимым из файла `ajax.php` (рис. 2.23).

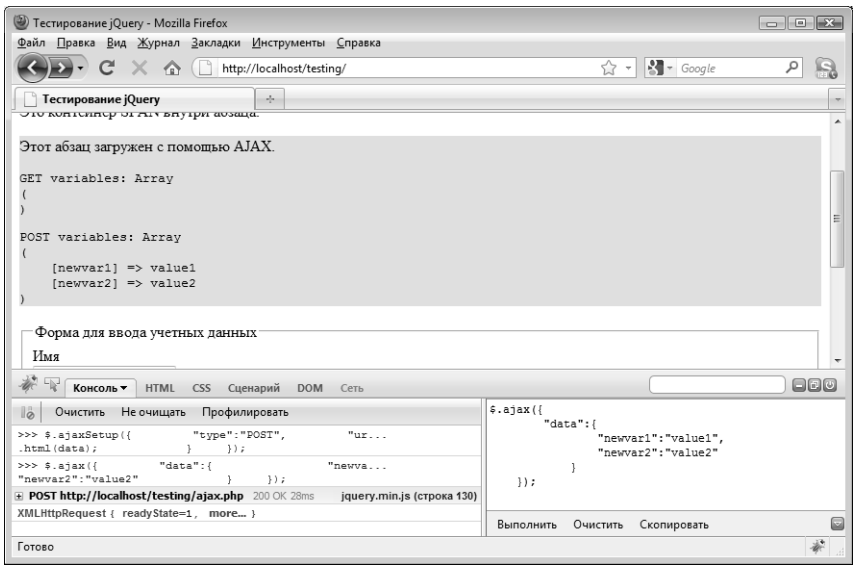



**Рис. 2.23.** Результат вызова AJAX после установки значений параметров по умолчанию

Установленные значения по умолчанию можно заменить в последующих вызовах `$.ajax()` простым переопределением значения параметра.

```
$.ajax({
        "type":"GET",
        "data":{
                "newvar1":"value3",
                "newvar2":"value4"
            }
});
```

Это приведет к отправке данных посредством процедуры GET (рис. 2.24).

## Использование прямых методов AJAX

Существует несколько простых методов целевого использования, доступных для решения обычных задач AJAX. По сути, эти прямые методы представляют собой оболочки, служащие исключительно для вызова функции `$.ajax()` с заранее установленными значениями ряда параметров.

Использование этих методов влечет за собой незначительную потерю производительности, поскольку при этом вы фактически вызываете метод, внутри которого происходит установка значений параметров и осуществляется вызов собственно функции `$.ajax()`. Однако удобства, обеспечиваемые использованием прямых методов, в действительности лишь ускоряют разработку многих сценариев.

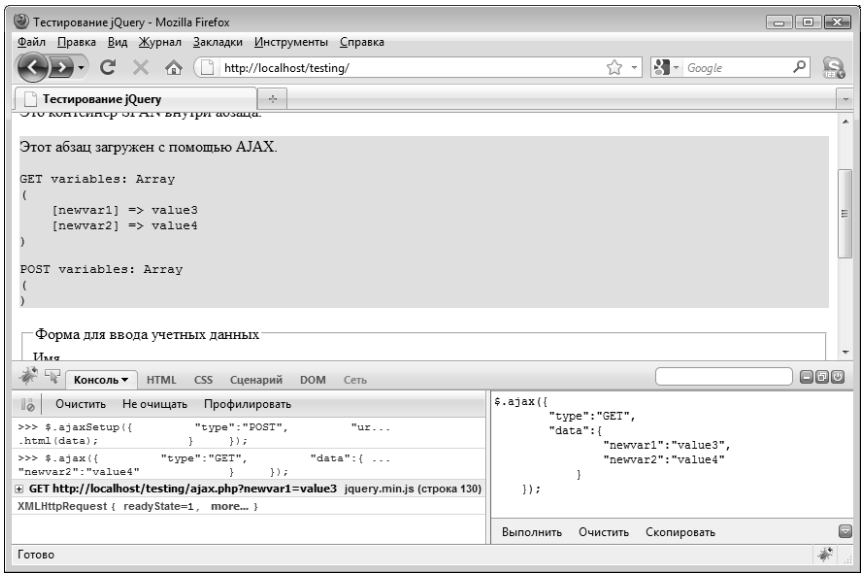



**Рис. 2.24.** Вид результата после замены процедуры отправки, установленной по умолчанию, процедурой GET

### `$.get()` и `$.post()`

Для стандартных запросов GET и POST предусмотрены простые функции `$.get()` и `$.post()`. Обе функции принимают по четыре аргумента: адрес URL, по которому отправляется запрос; необязательные данные, подлежащие передаче удаленному сценарию; необязательную функцию обратного вызова, которая выполняется в случае успешного завершения запроса; и необязательный параметр `dataType`.

Загрузите результат работы сценария `ajax.php` по процедуре GET без пересылки данных, используя для этого следующий код.

```
$.get("ajax.php", function(data){
        $("#bar")
            .css("background","yellow")
            .html(data);
    });
```

Для отправки запроса с данными по процедуре POST вполне подошел бы следующий код.

```
$.post("ajax.php", {"var1":"value"}, function(data){
        $("#bar")
            .css("background","yellow")
            .html(data);
    });
```

### `$.getJSON()`

Функция `$.getJSON()` служит прямым методом для загрузки данных в формате JSON. Ее аргументами являются адрес URL, по которому отправляется запрос, необязательные данные и необязательная функция обратного вызова. Для запуска

соответствующего примера вам понадобится еще один тестовый файл. С этой целью создайте в папке `testing` новый файл с именем `json.php` и введите в него следующие данные в формате JSON:

```
{"var1":"value1","var2":"value2"}
```

После этого загрузите содержимое файла `json.php` и выведите содержимое абзаца с идентификатором `bar`.

```
$.getJSON("json.php",
        function(data){
                $("#bar")
                    .css("background","yellow")
                    .html(data.var1+","+data.var2);
            });
```

В результате выполнения этого кода содержимое указанного абзаца будет заменено строкой `"value1, value2"`.

### $.getScript()

Для загрузки внешних сценариев JavaScript используется функция `$.getScript()`. Ее аргументами являются адрес URL, по которому отправляется запрос, и необязательная функция обратного вызова (необходимость в которой обычно отсутствует, поскольку в случае успешной загрузки автоматически выполнится сценарий).

Создайте в папке `testing` новый файл `script.php` и введите в него следующий текст:

```
alert("Этот сценарий был загружен AJAX!");
```

А теперь загрузите этот файл, выполнив в консоли приведенный ниже код:

```
$.getScript("script.php");
```

В результате на экране отобразится окно с соответствующим сообщением.

### .load()

Метод `.load()` работает аналогично методам `$.get()` и `$.post()`, за исключением того, что он является именно методом, а не глобальной функцией. У него имеется неявная функция обратного вызова, которая заменяет HTML-код в элементах, отобранных селектором, содержимым, возвращенным из удаленного файла.

Данный метод принимает те же самые три аргумента: целевой адрес URL, необязательные данные и необязательную функцию обратного вызова (которая запускается после замены содержимого в элементе). Загрузите содержимое файла `ajax.php` в абзац с идентификатором `bar` после отправки некоторых данных, используя следующий код:

```
$("#bar").load("ajax.php", {"var1":"value1"});
```

После выполнения этого кода содержимое абзаца будет заменено возвращенным результатом.

# Резюме

В этой необычайно насыщенной главе было рассмотрено множество очень важных вопросов. Обязательно ознакомьтесь с документацией jQuery API, где можно найти дополнительные примеры и описания функций, а также обсуждение затронутых тем в сообществе разработчиков. Чтобы найти описание метода, достаточно

добавить его название в конце URL-адреса справки API. Например, если вас интересует метод `.slideup()`, введите в адресной строке браузера такой адрес: `http://api.jquery.com/slideup`.

Следующая часть книги поможет вам освежить свои знания по PHP, включая объектно-ориентированное программирование, после чего мы примемся за разработку календаря событий, который начнем создавать в главе 4.

# Часть II

# Профессиональные аспекты программирования на PHP

А сейчас мы отвлечемся на некоторое время от jQuery и сосредоточим свое внимание на вопросах, связанных с использованием PHP в серверных сценариях. В этой части вы узнаете о том, как планировать и реализовывать объектно-ориентированные серверные решения. Рассмотрение будет проводиться на примере создания календаря событий, который мы впоследствии улучшим, используя возможности jQuery. Далее будет предполагаться, что с базовыми понятиями PHP (переменные, функции, основные конструкции языка) вы достаточно хорошо знакомы, но если вам потребуется освежить свои знания основ PHP, можете обратиться к книге *Освой самостоятельно PHP за 24 часа 3-е издание* (Вильямс, 2008 г.).

# Глава 3

# Объектно-ориентированное программирование

В этой главе вы познакомитесь с основными концепциями **объектно-ориентированного программирования** (ООП) — стиля написания программ, в котором тесно связанные между собой операции и данные группируются в классы для повышения компактности и эффективности создаваемого кода.

Серверная часть проекта, который будет создаваться в этой книге, в значительной степени основана на идеях ООП, поэтому при рассмотрении всех последующих упражнений мы будем часто ссылаться на материал данной главы.

## Принципы ООП

Как уже было отмечено, объектно-ориентированное программирование — это стиль написания программ, позволяющий разработчику группировать родственные задачи в классы. Это обеспечивает легкость сопровождения кода и его соответствие принципам DRY (англ. "Don't Repeat Yourself" — не повторяйся).

**Примечание.** Более подробные сведения о принципах DRY-программирования можно найти по следующему адресу:
`http://en.wikipedia.org/wiki/Don't_repeat_yourself`.

Одно из главных достоинств DRY-подхода состоит в том, что обновление программ, спроектированных в этом стиле, требует внесения изменений лишь в определенных местах кода. Поддержка программ с объявлениями переменных, разбросанными по всему коду, превращается для разработчиков в сущий кошмар, ведь в этом случае они вынуждены многократно просматривать весь текст программы, чтобы в процессе ее изменения не допустить дублирования данных или функциональности.

Многие программисты испытывают определенный страх перед объектно-ориентированным программированием, поскольку такой подход требует применения нового синтаксиса, который на первый взгляд представляется более сложным по сравнению с синтаксисом **процедурного**, или макрокомандного, программирования. Однако более тщательный анализ показывает, что код, написанный с использованием ООП, в действительности отличается простотой и предельным рационализмом.

# Объекты и классы

Прежде чем углубляться в изучение тонкостей ООП, вам следует ознакомиться с базовыми элементами объектов и классов. В этом разделе изучаются основные строительные блоки классов, их возможности и способы использования.

## Различия между объектами и классами

Сразу же отметим, что при обсуждении ООП иногда возникает определенная путаница: если послушать разговор опытных разработчиков, касающийся объектов и классов, то может сложиться впечатление, что эти термины взаимозаменяемы. Однако это далеко не так.

Если обратиться к аналогии, то **класс** можно уподобить плану дома. План отображает схему строения дома на бумаге, точно определяя взаимосвязь его отдельных частей между собой, даже если дом еще не существует.

Тогда **объект** можно уподобить самому дому, построенному в соответствии с планом. Данные, хранящиеся в объекте, можно сравнить с древесиной, проводкой или бетоном, входящими в конструкцию дома: не будучи собранными в соответствии с планом, они остаются всего лишь грудой стройматериалов. В то же время, если соединить их в единое целое в соответствии с планом, возникнет полезным образом организованная конструкция — собственно дом.

Классы образуют структуру, состоящую из данных и операций, и эта информация используется для построения объектов. На основе одного и того же класса может быть создано несколько совершенно независимых объектов. Если продолжить нашу строительную аналогию, то создание ряда объектов одного класса можно сравнить с возведением серии домов на основании одного и того же плана: скажем, 150 различных домов, которые выглядят одинаково, но каждый снабжен собственной адресной табличкой и отличается свойственной только ему внутренней отделкой.

## Структура классов

Первое, что необходимо сделать при построении класса, — это объявить его. Соответствующий синтаксис довольно прост: сначала записывается ключевое слово `class`, за ним — имя, присвоенное классу, а в конце — пара фигурных скобок (`{}`), внутри которых размещаются определения элементов, составляющих класс.

```
<?php

class MyClass
{
    // Сюда помещаются свойства и методы класса
}

?>
```

Имея подобное объявление, можно создать экземпляр нового класса и сохранить его в переменной, используя ключевое слово `new`:

```
$obj = new MyClass;
```

Для просмотра содержимого класса можно воспользоваться функцией `var_dump()`:

```
var_dump($obj);
```

Протестируйте это на практике, поместив весь предыдущий код в файл с именем `test.php`, который следует предварительно создать в папке `testing`.

```
<?php

class MyClass
{
    // Сюда помещаются свойства и методы класса
}

$obj = new MyClass;
var_dump($obj);

?>
```

Загрузив в браузере страницу `http://localhost/testing/test.php`, вы получите следующий результат.

```
object(MyClass)#1 (0) { }
```

## Определение свойств класса

Для добавления данных в класс используются **свойства**, представляющие собой переменные, специфические для данного класса. Свойства работают аналогично обычным переменным, но, в отличие от них, жестко связаны с объектом, и поэтому для доступа к ним должен использоваться сам объект.

Чтобы добавить свойство в класс `MyClass`, включите в сценарий код, выделенный ниже полужирным шрифтом.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";
}

$obj = new MyClass;
var_dump($obj);

?>
```

Ключевое слово `public` определяет область видимости свойства, о чем будет подробнее говориться далее. Свойству присвоено имя в соответствии с обычными правилами именования переменных, а также значение (хотя присваивать значения свойствам класса вовсе *не обязательно*).

Чтобы получить значение свойства и вывести его в браузере, необходимо сослаться на соответствующий объект и на само свойство.

```
echo $obj->prop1;
```

Одновременно могут существовать несколько экземпляров одного и того же класса, поэтому отсутствие ссылки на конкретный объект приведет к тому, что сценарий не сможет распознать источник, из которого должна быть считана информация. В ООП для доступа к свойствам и методам, принадлежащим данному объекту, используется синтаксическая конструкция, включающая в себя стрелку (`->`).

Внесите в сценарий, хранящийся в файле `test.php`, изменения, выделенные в приведенном ниже коде полужирным шрифтом, что позволит прочитать свойства класса без вывода остального его содержимого.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";
}

$obj = new MyClass;
echo $obj->prop1

?>
```

Перезагрузив браузер, вы получите следующий результат.

```
Это свойство класса!
```

## Определение методов класса

**Методы** — это функции, специфические для класса. Отдельные операции, которые объект способен выполнять, определяются в классе в виде методов.

Например, чтобы создать методы, позволяющие устанавливать и получать значение свойства `$prop1`, добавьте в код строки, выделенные ниже полужирным шрифтом.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

$obj = new MyClass;
echo $obj->prop1

?>
```

**Примечание.** В ООП объекты могут ссылаться на самих себя с помощью конструкции `$this`. Работая с методом, используйте выражение `$this` точно так же, как использовали бы имя объекта вне класса.

Чтобы воспользоваться методом, следует вызвать его, как обычную функцию, дополнительно сославшись на объект, которому он принадлежит. Получите значение

свойства $prop1 класса MyClass, а затем установите и выведите новое значение этого свойства, внеся в сценарий изменения, выделенные ниже полужирным шрифтом.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

$obj = new MyClass;
// Получить значение свойства
echo $obj->getProperty();

// Установить новое значение
$obj->setProperty("Это новое значение свойства!");

// Вновь прочитать свойство для демонстрации изменений
echo $obj->getProperty();
?>
```

Перезагрузив браузер, вы получите следующий результат.

```
Это свойство класса!
Это новое значение свойства!
```

Преимущества ООП становятся очевидными, когда используются несколько экземпляров одного и того же класса. Добавьте в код еще один экземпляр класса MyClass и немного поэкспериментируйте с установкой и извлечением значений свойств.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

// Создать два объекта
```

```
$obj = new MyClass;
$obj2 = new MyClass;

// Получить значения свойства $prop1 для обоих объектов
echo $obj->getProperty();
echo $obj2->getProperty();

// Установить новые значения свойств для обоих объектов
$obj->setProperty("Это новое значение свойства!");
$obj->setProperty("Это свойство принадлежит второму экземпляру!");

// Вывести значения свойства $prop1 для обоих объектов
echo $obj->getProperty();
echo $obj2->getProperty();

?>
```

Перезагрузив браузер, вы получите следующий результат.

```
Это свойство класса!
Это свойство класса!
Это новое значение свойства!
Это свойство принадлежит второму экземпляру!
```

Нетрудно заметить, что ООП поддерживает объекты как независимые сущности, что облегчает разделение программы на небольшие, взаимодействующие между собой фрагменты кода.

Чтобы еще более упростить использование объектов, PHP предоставляет ряд так называемых **магических методов**, т.е. специальных методов, которые вызываются при выполнении некоторых типичных операций внутри объектов. Это позволяет разработчикам сравнительно легко решать множество полезных задач.

### *Конструкторы и деструкторы*

Во многих случаях некоторые свойства объектов желательно устанавливать уже в процессе создания объекта. Для этих целей PHP предоставляет магический метод `__construct`, называемый *конструктором*, который автоматически вызывается всякий раз при создании нового объекта данного класса.

Чтобы продемонстрировать концепцию конструкторов, добавим в класс `MyClass` конструктор, выводящий сообщение всякий раз, когда создается новый экземпляр.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }
```

```
    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}
// Создать новый объект
$obj = new MyClass;

// Получить значение свойства $prop1
echo $obj->getProperty();

// Вывести сообщение в конце файла
echo "Конец файла.<br />";

?>
```

**Примечание.** Здесь __CLASS__ является так называемой **магической константой**, в данном случае возвращающей имя класса, в котором она вызывается. Существует ряд магических констант, о которых можно подробнее прочитать в руководстве по PHP, находящемся по следующему адресу: http://us3.php.net/manual/en/language.constants.predefined.php

После перезагрузки этого файла в браузере отобразится следующий результат.

```
Инициирован класс "MyClass"!
Это свойство класса!
Конец файла!
```

Для вызова функций во время уничтожения объектов предусмотрен магический метод __destruct(), получивший название *деструктора*. Этот метод целесообразно использовать для освобождения ресурсов, принадлежащих классу (например, для закрытия соединения с базой данных).

Организуем вывод сообщения при уничтожении объекта, определив в классе MyClass магический метод __destruct().

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }
    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
```

```
    {
        return $this->prop1 . "<br />";
    }
}

// Создать новый объект
$obj = new MyClass;

// Получить значение свойства $prop1
echo $obj->getProperty();

// Вывести сообщение в конце файла
echo "Конец файла.<br />";

?>
```

Перезагрузка файла, в котором определен деструктор, приведет к следующему результату.

```
Инициирован класс "MyClass"!
Это свойство класса!
Конец файла!
Уничтожен класс "MyClass"!
```

Когда достигается конец файла, PHP автоматически освобождает все ресурсы, использовавшиеся в сценарии, чтобы не занимать лишний объем памяти. При этом автоматически запускается деструктор объекта `MyClass`.

Деструктор можно запустить явно, уничтожив объект с помощью функции `unset()`.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
```

```
}

// Создать новый объект
$obj = new MyClass;

// Получить значение свойства $prop1
echo $obj->getProperty();

// Уничтожить объект
unset($obj);

// Вывести сообщение в конце файла
echo "Конец файла.<br />";

?>
```

Теперь результат, который отобразится после перезагрузки браузера, будет следующим.

```
Инициирован класс "MyClass"!
Это свойство класса!
Уничтожен класс "MyClass"!
Конец файла!
```

### Преобразование объекта в строку

Для вывода объекта в виде строки предусмотрен еще один магический метод: `__toString()`.

Попытки вывести объект в виде строки, минуя метод `__toString()`, приведут к возникновению фатальной ошибки. Попытаемся, например, использовать команду `echo` для вывода объекта без привлечения магического метода.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function setProperty($newval)
    {

        $this->prop1 = $newval;
    }

    public function getProperty()
```

```
    {
        return $this->prop1 . "<br />";
    }
}

// Создать новый объект
$obj = new MyClass;

// Вывести объект в виде строки
echo $obj;

// Уничтожить объект
unset($obj);

// Вывести сообщение в конце файла
echo "Конец файла.<br />";

?>
```

Результат работы этого сценария будет следующим.

```
Инициирован класс "MyClass"!

Catchable fatal error: Object of class MyClass could not
be converted to string in
C:\XAMPP\xampp\htdocs\testing\test.php on line 33
```

Чтобы избежать этой ошибки, добавьте в сценарий метод `__toString()`.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }
```

```
    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

// Создать новый объект
$obj = new MyClass;

// Вывести объект в виде строки
echo $obj;

// Уничтожить объект
unset($obj);

// Вывести сообщение в конце файла
echo "Конец файла.<br />";

?>
```

В этом случае при попытке преобразовать объект в строку будет вызван метод `getProperty()`. Перезагрузив тестовый сценарий в браузере, вы получите следующий результат.

```
Инициирован класс "MyClass"!
Используем метод toString: Это свойство класса!
Уничтожен класс "MyClass"!
Конец файла!
```

**Совет.** Существует ряд других магических методов, кроме тех, которые обсуждались в этом разделе. Полный список магических методов приводится в руководстве по PHP, которое находится по следующему адресу: `http://us2.php.net/manual/en/language.oop5.magic.php`

## Наследование классов

Классы могут наследовать методы и свойства другого класса с помощью ключевого слова `extends`. Чтобы создать класс, который расширяет класс `MyClass` и вводит новый метод, внесите в тестовый файл следующие изменения.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }
```

```
    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

class MyOtherClass extends MyClass
{
    public function newMethod()
    {
        echo 'Новый метод в классе "', __CLASS__, '".<br />';
    }
}

// Создать новый объект
$newobj = new MyOtherClass;

// Вывести объект в виде строки
echo $newobj->newMethod();

// Использовать метод родительского класса
echo $newobj->getProperty();

?>
```

Перезагрузив тестовый сценарий в браузере, вы получите следующий результат.

```
Инициирован класс "MyClass"!
Новый метод в классе "MyOtherClass".
Это свойство класса!
Уничтожен класс "MyClass"!
```

### Переопределение унаследованных свойств и методов

Чтобы изменить поведение существующего свойства или метода в новом классе, достаточно переопределить его путем повторного объявления в новом классе.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
```

```
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        echo 'Новый конструктор в классе "', __CLASS__, '".<br />';
    }

    public function newMethod()
    {
        echo 'Новый метод в классе "', __CLASS__, '".<br />';
    }
}

// Создать новый объект
$newobj = new MyOtherClass;

// Вывести объект в виде строки
echo $newobj->newMethod();

// Использовать метод родительского класса
echo $newobj->getProperty();

?>
```

Этот сценарий выведет в браузере следующий результат.

```
Новый конструктор в классе "MyOtherClass".
Новый метод в классе "MyOtherClass".
Это свойство класса!
Уничтожен класс "MyClass"!
```

### *Сохранение функциональности исходного метода при переопределении методов*

Если вы добавили новую функциональность в унаследованный метод, но хотите обратиться к исходному методу, используйте ключевое слово `parent` с **оператором разрешения области видимости** (`::`).

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        // Вызвать конструктор родительского класса
        parent::__construct();
        echo 'Новый конструктор в классе "', __CLASS__, '".<br />';
    }

    public function newMethod()
    {
        echo 'Новый метод в классе "', __CLASS__, '".<br />';
    }
}
```

```
// Создать новый объект
$newobj = new MyOtherClass;

// Вывести объект в виде строки
echo $newobj->newMethod();

// Использовать метод родительского класса
echo $newobj->getProperty();

?>
```

Этот сценарий выводит результаты работы как родительского конструктора, так и конструктора нового класса.

```
Инициирован класс "MyClass"!
Новый конструктор в классе "MyOtherClass".
Новый метод в классе "MyOtherClass".
Это свойство класса!
Уничтожен класс "MyClass"!
```

## Назначение области видимости свойствам и методам

Чтобы обеспечить дополнительный контроль над объектами, методами и свойствами, им присваивается **видимость**. Эта характеристика позволяет контролировать, откуда именно к свойствам, методам и объектам можно получать доступ. Существуют три области видимости, определяемые соответствующими ключевыми словами: **общедоступный** (`public`), **защищенный** (`protected`) и **закрытый** (`private`). В дополнение к видимости, метод и свойство могут объявляться **статическими** (`static`), что позволяет обращаться к ним без предварительного создания экземпляра класса.

**Примечание.** Контроль доступа с помощью области видимости был впервые введен в PHP 5. Информацию относительно PHP 4 можно найти в руководстве по PHP по следующему адресу: `http://us2.php.net/manual/en/language.oop5.php`

### *Общедоступные свойства и методы*

Все методы и свойства, которые мы до сих пор использовали, были общедоступными (`public`). Это означает, что доступ к ним был разрешен из любого места сценария, как изнутри, так и извне класса.

### *Защищенные свойства и методы*

Если свойство или метод объявлены как защищенные (`protected`), то это означает, что доступ к ним возможен только изнутри самого класса или его классов-потомков (классов, которые унаследованы от класса, содержащего защищенный метод).

Объявите метод `getProperty()` как защищенный в классе `MyClass` и попытайтесь получить к нему доступ извне класса.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";
```

```
    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    protected function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        parent::__construct();
        echo 'Новый конструктор в классе "', __CLASS__, '".<br />';
    }

    public function newMethod()
    {
        echo 'Новый метод в классе "', __CLASS__, '".<br />';
    }
}

// Создать новый объект
$newobj = new MyOtherClass;

// Попытаться вызвать защищенный метод
echo $newobj->getProperty();

?>
```

Попытка выполнить этот сценарий приведет к ошибке.

```
Инициирован класс "MyClass"!
Новый конструктор в классе "MyOtherClass".


Fatal error: Call to protected method
```

```
MyClass::getProperty() from context '' in
C:\XAMPP\xampp\htdocs\testing\test.php on line 53
```

А теперь создайте в классе `MyOtherClass` новый метод и используйте его для вызова метода `getProperty()`.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    protected function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        // Вызвать конструктор родительского класса

        parent::__construct();
        echo 'Новый конструктор в классе "', __CLASS__, '".<br />';
    }

    public function newMethod()
    {
        echo 'Новый метод в классе "', __CLASS__, '".<br />';
    }

    public function callProtected()
```

```
    {
        return $this->getProperty();
    }
}

// Создать новый объект
$newobj = new MyOtherClass;

// Вызвать защищенный метод из общедоступного метода
echo $newobj->callPropected();

?>
```

Теперь этот сценарий сгенерирует требуемый результат.

```
Инициирован класс "MyClass"!
Новый конструктор в классе "MyOtherClass".
Это свойство класса!
Уничтожен класс "MyClass"!
```

### Закрытые свойства и методы

Свойство или метод, объявленные как закрытые (`private`), доступны лишь в классе, в котором они определены. Это означает, что даже если новый класс расширяет класс, содержащий определение закрытого свойства, это свойство или метод не будут доступны в дочернем классе.

Чтобы это продемонстрировать, объявим метод `getProperty()` в классе `MyClass` как закрытый и попытаемся вызвать метод `callProtected()` из класса `MyOtherClass`.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }
```

```
    private function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        // Вызвать конструктор родительского класса
        parent::__construct();
        echo 'Новый конструктор в классе "', __CLASS__, '".<br />';
    }

    public function newMethod()
    {
        echo 'Новый метод в классе "', __CLASS__, '".<br />';
    }

    public function callProtected()
    {
        return $this->getProperty();
    }
}

// Создать новый объект
$newobj = new MyOtherClass;

// Использовать метод из родительского класса
echo $newobj->callProtected();

?>
```

После перезагрузки браузера вы столкнетесь с такой ошибкой.

```
Инициирован класс "MyClass"!
Новый конструктор в классе "MyOtherClass".


Fatal error: Call to private method
MyClass::getProperty() from context 'MyOtherClass' in
C:\XAMPP\xampp\htdocs\testing\test.php on line 50
```

### Статические свойства и методы

К методам или свойствам, объявленным как статические (`static`), можно получить доступ без обязательного предварительного создания экземпляра класса; вы просто предоставляете имя класса, дополняете его оператором разрешения области видимости и указываете имя свойства или метода.

Одно из главных преимуществ использования статических свойств состоит в том, что содержащиеся в них значения хранятся на протяжении всего времени жизни сценария. Это означает, что если вы измените статическое свойство, а затем обратитесь к нему из сценария позднее, то по-прежнему обнаружите там сохраненное значение.

Чтобы продемонстрировать это, добавим в класс `MyClass` статическое свойство `$count` и статический метод `plusOne()`. Организуем также цикл `do...while` для вывода увеличивающегося значения счетчика `$count`, пока оно не превышает `10`.

```
<?php

class MyClass
{
    public $prop1 = "Это свойство класса!";

    public static $count = 0;

    public function __construct()
    {
        echo 'Инициирован класс "', __CLASS__, '"!<br />';
    }

    public function __destruct()
    {
        echo 'Уничтожен класс "', __CLASS__, '"!<br />';
    }

    public function __toString()
    {
        echo "Используем метод toString: ";
        return $this->getProperty();
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    private function getProperty()
    {
        return $this->prop1 . "<br />";
    }

    public static function plusOne()
    {
        return "Значение счетчика: " . ++self::$count . ".<br />";
    }

}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        // Вызвать конструктор родительского класса
        parent::__construct();
        echo 'Новый конструктор в классе "', __CLASS__, '".<br />';
    }
```

```
    public function newMethod()
    {
        echo 'Новый метод в классе "', __CLASS__, '".<br />';
    }

    public function callProtected()
    {
        return $this->getProperty();
    }
}

do
{
    // Вызвать plusOne без создания экземпляра MyClass
    echo MyClass::plusOne();
} while (MyClass::$count < 10);

?>
```

**Примечание.** При обращении к статическим свойствам знак доллара ($) ставится *после* оператора разрешения области видимости.

Загрузив этот сценарий в браузер, вы получите следующий результат.

```
Значение счетчика: 1
Значение счетчика: 2
Значение счетчика: 3
Значение счетчика: 4
Значение счетчика: 5
Значение счетчика: 6
Значение счетчика: 7
Значение счетчика: 8
Значение счетчика: 9
Значение счетчика: 10
```

## Создание комментариев в стиле Doc-блоков

Хотя создание комментариев в стиле Doc-блоков (DocBlock) и не является формальной частью ООП, этот метод широко используется при документировании классов. Помимо того что он служит своего рода стандартом, которым разработчики могут руководствоваться при написании кода, он уже внедрен в наиболее популярные SDK (Software Development Kit — набор средств для разработки ПО), такие как Eclipse (`http://eclipse.org/`) и NetBeans (`http://netbeans.org/`), и будет использоваться для генерации сопроводительных комментариев к коду.

Doc-блоки определяются путем использования многострочных комментариев, в начале строк которых содержится дополнительная "звездочка".

```
/**
 * Это простейший Doc-блок
 */
```

Удобство Doc-блоков обусловлено возможностью использования меток, начинающихся символом `@`, сразу за которым следует имя метки и ее значение. Это позволяет разработчику указать имя автора сценария, лицензию для класса, информацию о свойстве или методе и любую другую полезную информацию.

Ниже приводится список наиболее часто используемых меток.

- **@author.** Эта метка позволяет указать автора текущего элемента (каковым может быть класс, файл, метод или любой другой фрагмент кода). В одном и том же Doc-блоке можно использовать несколько меток `@author`, если автор не один. Формат использования: `John Doe <john.doe@email.com>`.
- **@copyright.** Позволяет указать год регистрации авторских прав и имя их владельца для текущего элемента. Формат использования: `2010` *имя_владельца_лицензии*.
- **@license.** Ссылка на лицензию для текущего элемента. Формат использования: `http://www.example.com/path/to/license.txt` *Название_лицензии*.
- **@var.** Хранит тип и описание переменной или свойства класса. Формат использования: *тип описание_элемента*.
- **@param.** Эта метка отображает тип и описание параметра функции или метода. Формат использования: *тип $имя_элемента описание_элемента*.
- **@return.** Эта метка предоставляет тип и описание значения, возвращаемого функцией или методом. Формат задания информации для этой метки следующий: *тип описание_возвращаемого_элемента*.

Ниже показан пример включения комментариев в класс с использованием Doc-блоков.

```
<?php

/**
 * Простой класс
 *
 * Это расширенное описание данного класса, которое может
 * содержать сколько угодно строк. Оно не является обязательным,
 * тогда как краткое описание требуется всегда.
 *
 * Кроме того, это описание может состоять из нескольких
 * абзацев, если такое многословие оправдано.
 *
 * @author Jason Lengstorf <jason.lengstorf@ennuidesign.com>
 * @copyright 2010 Ennui Design
 * @license http://www.php.net/license/3_01.txt PHP License 3.01
 */
class SimpleClass
{
    /**
     *
     * Общедоступная переменная
     *
     * @var string: хранит данные для класса
     */
     public $foo;
    /**
     * Устанавливает для $foo новое значение
     * при создании экземпляра класса
     *
     * @param string $val: параметр класса
     * @return void
```

```
     */
    public function __construct($val)
    {
        $this->foo = $val;
    }

    /**
     * Перемножает два целых числа
     *
     * Принимает два целых числа и возвращает их
     * произведение
     *
     * @param int $bat: сомножитель
     * @param int $baz: сомножитель
     * @return int: произведение двух параметров
     */
    public function bar($bat, $baz)
    {
        return $bat * $baz;
    }
}

?>
```

Даже беглого просмотра этого класса достаточно для того, чтобы преимущества Doc-блоков стали очевидными: в коде отсутствуют неясности, так что тот, кому придется с ним работать, сможет сразу взяться за дело, не теряя времени на догадки относительно того, какую роль играет тот или иной фрагмент кода.

---

**Примечание.** Более подробную информацию относительно Doc-блоков можно найти по такому адресу:
`http://ru.wikipedia.org/wiki/PHPDoc`

---

# Преимущества ООП в сравнении с процедурным подходом

Вообще-то подходы к написанию программных кодов нельзя делить на "правильные" и "неправильные". И все же в этом разделе приводятся весьма веские аргументы в пользу внедрения объектно-ориентированного подхода при разработке программного обеспечения, особенно в случае крупных приложений.

## Простота реализации

Хотя поначалу принципы ООП могут немного смущать, на самом деле это более простой подход к работе с данными. Поскольку объект может хранить данные внутри себя, для правильной работы переменных вовсе не обязательно передавать их от функции к функции.

Кроме того, поскольку одновременно может существовать несколько экземпляров одного и того же класса, обработка больших наборов данных значительно упрощается. Предположим, например, что необходимо организовать обработку информации, касающейся двух лиц. Соответствующими данными будут служить их имена, профессии и возраст.

### Процедурный подход

Ниже приведен код для нашего примера, написанный с использованием процедурного подхода.

```
<?php

function changeJob($person, $newjob)
{
// Изменить профессию
    $person['job'] = $newjob;
    return $person;
}
function happyBirthday($person)
{
// Прибавить 1 к возрасту
    ++$person['age'];
    return $person;
}

$person1 = array(
    'name' => 'Tom',
    'job' => 'Button-Pusher',
    'age' => 34
);

$person2 = array(
    'name' => 'John',
    'job' => 'Lever-Puller',
    'age' => 41
);

// Вывести начальные значения характеристик
echo "<pre>Person 1: ", print_r($person1, TRUE), "</pre>";
echo "<pre>Person 2: ", print_r($person2, TRUE), "</pre>";

// Том получил повышение, и у него был день рождения
$person1 = changeJob($person1, 'Box-Mover');
$person1 = happyBirthday($person1);

// У Джона был только день рождения
$person2 = happyBirthday($person2);

// Вывести новые значения характеристик
echo "<pre> Person 1: ", print_r($person1, TRUE), "</pre>";
echo "<pre> Person 2: ", print_r($person2, TRUE), "</pre>";
?>
```

Выполнив этот код, вы получите следующие результаты.

```
Person 1: Array
(
    [name] => Tom
    [job] => Button-Pusher
    [age] => 34
)
```

```
Person 2: Array
(
    [name] => John
    [job] => Lever-Puller
    [age] => 41
)
Person 1: Array
(
    [name] => Tom
    [job] => Box-Mover
    [age] => 35
)
Person 2: Array
(
    [name] => John
    [job] => Lever-Puller
    [age] => 42
)
```

Хотя этот код не столь уж и плох, в процессе его написания необходимо помнить о множестве вещей. При каждом вызове функции в нее должен передаваться, а затем и возвращаться из нее, массив, содержащий характеристики соответствующего лица, и здесь легко допустить ошибку.

Для оптимизации этого кода было бы желательно, чтобы разработчик оперировал как можно меньшим количеством элементов. Каждый раз для выполнения текущей операции в функцию должна передаваться лишь та информация, без которой невозможно обойтись.

Для обеспечения именно такого структурирования кода и предназначено ООП.

### Подход ООП

Ниже приведен тот же пример, но реализованный в стиле ООП.

```
<?php

class Person
{
    private $_name;
    private $_job;
    private $_age;

    public function __construct($name, $job, $age)
    {
        $this->_name = $name;
        $this->_job = $job;
        $this->_age = $age;
    }

    public function changeJob($newjob)
    {
        $this->_job = $newjob;
    }

    public function happyBirthday()
    {
```

```
        ++$this->_age;
    }
}

// Создать два новых объекта
$person1 = new Person("Том", "Button-Pusher", 34);
$person2 = new Person("Джон", "Lever Puller", 41);

// Вывести их начальные состояния
echo "<pre>Person 1: ", print_r($person1, TRUE), "</pre>";
echo "<pre>Person 2: ", print_r($person2, TRUE), "</pre>";

// Повысить Тома в должности и учесть изменение возраста
$person1->changeJob("Box-Mover");
$person1->happyBirthday();

// Джон лишь становится на год старше
$person2->happyBirthday();

// Вывести конечные результаты
echo "<pre>Person 1: ", print_r($person1, TRUE), "</pre>";
echo "<pre>Person 2: ", print_r($person2, TRUE), "</pre>";
?>
```

На этот раз в браузере будет выведена следующая информация.

```
Person 1: Person Object
(
    [_name:private] => Том
    [_job:private] => Button-Pusher
    [_age:private] => 34
)

Person 2: Person Object
(
    [_name:private] => Джон
    [_job:private] => Lever Puller
    [_age:private] => 41
)

Person 1: Person Object
(
    [_name:private] => Том
    [_job:private] => Box-Mover
    [_age:private] => 35
)

Person 2: Person Object
(
    [_name:private] => Джон
    [_job:private] => Lever Puller
    [_age:private] => 42
)
```

Чтобы воплотить в жизнь объектно-ориентированный подход, потребовалось немного потрудиться, но после того, как класс определен, создание и изменение объектов, соответствующих разным лицам, становится сущим пустяком: информацию о лице больше не надо передавать методам или получать от них — каждому методу передается лишь необходимая информация, без которой он не может обойтись.

Возможно, при небольшом программном коде это различие не особенно заметно, но как только приложение начнет расти в объеме, ООП, при правильной реализации, значительно снизит нагрузку на программиста.

**Примечание.** Не все фрагменты программы нуждаются в применении ООП-подхода. Вовсе не обязательно облекать в класс какую-нибудь локальную функцию, которая обрабатывает небольшую порцию данных в одном конкретном месте приложения. Выбирая между объектно-ориентированным и процедурным подходами, руководствуйтесь здравым смыслом.

## Улучшение структуры приложения

Дополнительное преимущество, обеспечиваемое ООП, — это упрощение процесса подготовки программного пакета и его каталогизации. Как правило, каждый класс может храниться в отдельном файле, а при использовании продуманной системы присвоения имен доступ к классам осуществляется чрезвычайно просто.

Предположим, у вас имеется приложение, насчитывающее 150 классов, которые динамически вызываются через управляющий файл, находящийся в корневом каталоге файловой системы приложения. Все 150 классов именуются в соответствии с одной и той же схемой — `class.`*`имя_класса`*`.php` — и находятся в папке `inc` приложения.

Вместо того чтобы включать в управляющий файл все 150 классов или использовать какой-либо хитроумный способ включения файлов в код, целесообразно использовать функцию PHP `__autoload()`, позволяющую организовать динамическое, по мере их вызова, извлечение только тех файлов, которые действительно необходимы.

```
<?php
    function __autoload($class_name)
    {
        include_once 'inc/class.' . $class_name . '.inc.php';
    }
?>
```

Кроме того, размещение классов в отдельных файлах облегчает перенос кода и упрощает его использование в новых приложениях, поскольку избавляет от необходимости выполнения многочисленных операций копирования и вставки.

## Легкость сопровождения

Благодаря свойственной объектно-ориентированному коду компактности, вносить в него изменения обычно гораздо проще, чем в длинный, запутанный код типа “спагетти”, являющийся характерным признаком процедурного подхода. Если некоторому массиву информации присваивается новый атрибут, то (в неблагоприятных случаях) процедурная часть программы может потребовать просмотра всего кода и присвоения нового атрибута *каждой функции, которая использует данный массив*.

В случае же объектно-ориентированных приложений для подобного обновления программы придется всего лишь добавить новое свойство и методы для работы с данным свойством.

Многие из рассмотренных в этой главе преимуществ объектно-ориентированного подхода обусловлены его использованием в сочетании с принципами DRY-программирования. Вполне можно создать легкий в обслуживании процедурный код, не доводящий до бессонницы, и точно так же можно создать никуда не годный объектно-ориентированный код. В этой книге демонстрируется, как именно ООП-подход, усиленный соблюдением проверенных методик программирования, способствует созданию удобочитаемого и легкого в сопровождении кода.

# Резюме

К этому времени объектно-ориентированный стиль программирования уже не должен вызывать у вас робости. Все ядро серверной части календаря событий будет основано на ООП, поэтому оставшиеся неясными для вас детали будут далее тщательно изучаться, по мере того как изложенные в этой главе концепции будут воплощаться в жизнь в виде реального приложения.

# Глава 4

# Создание календаря событий

Теперь, когда вы освежили свои знания в области объектно-ориентированного программирования, можно начать разработку календаря событий — проекта, лежащего в основе данной книги. Мы приступим к этому прямо сейчас и постепенно будем добавлять в календарь все новые и новые функциональные возможности, используя для этого PHP и jQuery.

## Планирование приложения

Поскольку мы создаем проект, что называется, "с чистого листа", стоит сначала продумать план действий. Приложение будет использовать базу данных (MySQL), поэтому наш план должен состоять из двух частей: определения структуры базы данных (БД) и составления простейшей схемы приложения, которое будет обращаться к БД и манипулировать содержащимися в ней данными.

### Определение структуры базы данных

Создание приложения значительно упростится, если мы предварительно спланируем, какие данные собираемся хранить. От этого будет зависеть, какие именно операции должно выполнять приложение.

В простейшем варианте календаря событий вполне можно обойтись минимальным набором хранимых элементов информации, перечень которых приводится ниже.

- **event_id.** Целочисленный идентификатор с автоматически увеличивающимся значением, который однозначно определяет каждое событие.
- **event_title.** Название события.
- **event_desc.** Полное описание события.
- **event_start.** Дата и время начала события (в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`).
- **event_end.** Дата и время окончания события (в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`).

## Создание схемы класса

Далее мы должны составить примерную схему класса, который будет обрабатывать все операции, выполняемые в приложении над календарем событий. Назовем этот класс **`Calendar`**; его структура должна соответствовать следующей общей процедуре его использования.

- Создаем конструктор.
  - Проверяем наличие соединения с базой данных и при необходимости создаем его.
  - Устанавливаем следующие основные свойства класса: объект базы данных, используемая дата, просматриваемый месяц, просматриваемый год, количество дней в месяце и неделя, с которой начинается месяц.
- Генерируем HTML-разметку, создающую веб-форму для календаря событий.
  - Проверяем, не выполняется ли изменение или создание события.
  - Загружаем информацию о событии в форму, если его данные необходимо изменить.
- Сохраняем новые события в базе данных, проверяя и при необходимости корректируя входную информацию.
- Удаляем события из базы данных и подтверждаем удаление.
- Загружаем данные о событиях.
  - Загружаем информацию о событиях из базы данных.
  - Сохраняем каждое событие в виде массива, соответствующего заданному дню месяца.
- Выводим HTML-разметку с информацией из календаря: используя массив событий, просматриваем в цикле каждый день месяца и в необходимых случаях присоединяем к нему названия событий и их временные характеристики.
- Отображаем информацию о событиях в виде HTML-разметки: для каждого события получаем уникальный идентификатор (ID) и загружаем описание и временные характеристики.

## Планирование структуры папок приложения

По мере разработки приложения его структура будет все более усложняться, поэтому стоит сделать небольшую паузу и продумать, как лучше всего организовать файлы.

В целях безопасности все, что только можно, следует хранить вне **корневой веб-папки документа** (document root) и других общедоступных папок. В частности, это касается учетных записей пользователей для доступа к базе данных, ядра приложения и классов, которые в нем выполняются. Если корневая папка не содержит файлов с критически важной информацией, то злоумышленники не смогут просмотреть вашу файловую структуру, не проникнув предварительно на сам сервер, что и является одним из элементов хорошей практики обеспечения безопасности.

Прежде всего создадим две папки: папку `public`, в которой будут содержаться файлы, непосредственно доступные пользователям приложения, такие как файлы CSS, индексный файл `index.php` или файлы JavaScript, и папку `sys`, предназначенную для хранения файлов, доступ к которым ограничен, таких как учетные записи пользователей базы данных, классы и PHP-файлы ядра приложения.

### Общедоступные файлы

В качестве корневой веб-папки используем папку `public`. Именно эту папку сервер будет просматривать в первую очередь при обращении пользователя к URL-адресу приложения. На корневом уровне она будет содержать перечисленные ниже файлы, доступ к которым необходим пользователям для просмотра базы данных и манипулирования хранящимися в ней элементами информации.

- **`index.php`.** Это основной файл, который отображает дни месяца в календарном формате с указанием названий событий для дней, с которыми они связаны.
- **`view.php`.** При выполнении пользователем щелчка на названии события отображается эта страница, предоставляющая подробные сведения о событии.
- **`admin.php`.** Отображаемая на этой странице форма используется для создания или изменения событий.
- **`confirmdelete.php`.** Событие будет удалено лишь после того, как пользователь подтвердит свои намерения путем отправки формы, предоставляемой на этой странице.

Кроме перечисленных файлов папка `public` будет содержать подпапку `assets`, предназначенную для хранения дополнительных файлов, необходимых сайту. Эти файлы будут группироваться в соответствии с назначением, но в данном разделе мы будем иметь дело с четырьмя категориями таких файлов: файлы общего назначения, CSS-файлы, файлы JavaScript и файлы, обеспечивающие обработку форм.

Создайте внутри папки `assets` четыре подпапки: `common`, `css`, `inc` и `js`. В папке `common` будут храниться файлы, используемые на всех общедоступных страницах (а именно, файлы с HTML-разметкой для верхнего и нижнего колонтитулов страниц), в папке `css` — таблицы стилей, в папке `inc` — файлы для обработки входных данных, отправляемых серверу вместе с формой, и в папке `js` — файлы JavaScript.

### Файлы с ограниченным доступом

Папка `sys` будет содержать три подпапки: `class` — для всех файлов классов приложения (таких, как класс `Calendar`), `config` — для хранения конфигурационной информации приложения, например учетных записей пользователей базы данных, и `core` — для файлов, осуществляющих инициализацию приложения.

Подготовив все папки и создав файлы, мы получим хорошо организованную файловую структуру, которую впоследствии можно будет легко масштабировать (рис. 4.1).

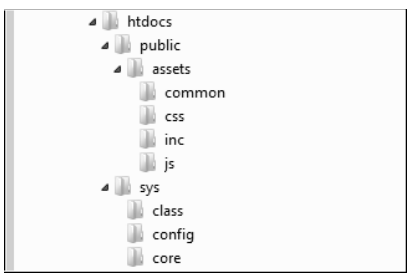



**Рис. 4.1.** Отображение структуры папок и файлов приложения в окне проводника

**Папки общедоступные и с ограниченным доступом — стоит ли вообще об этом заботиться?**

Возможно, вы зададите вопрос: "Зачем прилагать дополнительные усилия и разделять папки на общедоступные и с ограниченным доступом? Что это дает?"

Чтобы ответить на этот вопрос, следует сказать хотя бы пару слов о том, как работает сервер. В сущности, сервер — это компьютер, который хранит файлы и обеспечивает доступ к ним по сети (например, через Интернет) с использованием сетевого идентификатора (IP-адрес или URL, соответствующий IP-адресу). На одном сервере могут быть размещены сотни веб-сайтов или других приложений, каждое из которых хранится в собственной папке.

Сервер предоставляет внешним пользователям возможность обращения к общедоступным папкам, т.е. любой файл, находящийся в такой папке, доступен для пользователей сети, к которой подключен сервер. В случае файлов, содержащих критически важную информацию, это не всегда желательно.

К счастью, файлы, находящиеся в общедоступной папке, могут получать доступ к внешним по отношению к ней файлам, даже если пользователи сети такой возможности не имеют. Это позволяет скрывать важные данные от внешнего мира, оставляя их доступными для приложения.

Существуют и другие способы сокрытия подобной информации, но уже такая простая мера, как помещение критически важных файлов в папку с ограниченным доступом, частично решает указанную проблему.

## Настройка среды разработки

Поскольку приложение наряду с общедоступными папками использует также папки с ограниченным доступом, следует внести в среду разработки некоторые изменения, а именно: переадресовать сервер на общедоступную папку, а не на папку, содержащую подпапки обоих типов.

В этом разделе рассказано о том, как переадресовать сервер на папку `public`.

**Примечание.** Вы можете пропустить этот раздел и поместить папку `sys` в папку `public` без ущерба для функциональных возможностей приложения (просто в этом случае пути к файлам будут отличаться от тех, которые используются в примерах на протяжении всей книги). Однако, поступив таким образом, вы подвергнете безопасность приложения потенциальному риску. Поэтому все же крайне желательно, чтобы вы придерживались приведенных ниже рекомендаций.

### *Локальная разработка*

Чтобы изменить корневую папку документа (указать вместо нее папку `public`), потребуется внести изменения в конфигурационный файл сервера. В этой книге предполагается, что в качестве сервера, работающего совместно с пакетом XAMPP, используется сервер Apache, поэтому прежде всего вы должны перейти к месту расположения файла `httpd.conf` (соответствующие пути: `/xamppfiles/etc/httpd.conf` для Mac, `/opt/lampp/etc/httpd.conf` для Linux и `\xampp\apache\conf\httpd.conf` для Windows).

В файле `httpd.conf` найдите директиву `DocumentRoot`. Именно здесь вы установите путь к папке `public`. Измените текущий путь, чтобы он выглядел примерно так.

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory,
# but symbolic links and aliases may be used to point to other
# locations.
#
DocumentRoot "C:/XAMPP/xampp/htdocs/public"
```

Далее найдите в файле `httpd.conf` строку, содержащую ссылку на корневую папку документа, для установки разрешений. Приведите ее в соответствие с приведенной ниже строкой.

```
<Directory "C:/XAMPP/xampp/htdocs/public">
```

Найдя и изменив, как показано выше, пути к файлам, *перезапустите Apache*, воспользовавшись панелью управления XAMPP. С этого момента обращения к серверу будут по умолчанию переадресовываться на папку `public` приложения. Протестируйте это, создав файл `index.php` и добавив в него следующий код:

```
<?php echo "Это новая корневая папка документа!"; ?>
```

Перейдите в браузере к корневой веб-папке своей среды разработки (по умолчанию — `localhost`), чтобы удостовериться в правильности установленной конфигурации (рис. 4.2).

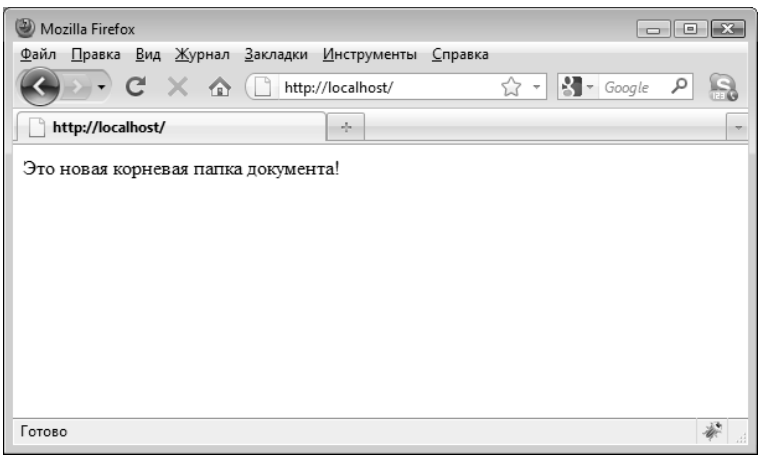



**Рис. 4.2.** Отображение файла `index.php` из папки `public` после изменения конфигурации сервера Apache

### *Удаленная разработка*

Поскольку для удаленной разработки обычно используют сервер компании, предоставляющей услуги хостинга, порядок переадресации вашего домена на папку `public` приложения зависит от провайдера, услугами которого вы пользуетесь, и поэтому в данной книге не рассматривается.

Однако во многих случаях хост-сервер разрешает переадресацию домена на папку в пределах своей учетной записи. Если это так, то достаточно переадресовать домен на папку `public`, и все будет работать нормально.

Некоторые хосты запрещают доступ за пределы корневой папки. Если ваш поставщик услуг хостинга использует именно такую политику безопасности, достаточно будет переместить папку `sys` в папку `public`, соответственно изменив пути доступа к файлам.

## Создание календаря

Подготовив структуру папок и настроив рабочую среду, можно приступать к фактической разработке календаря. Наш календарь будет иметь три различных пред-

ставления — основное, детальное (отображающее информацию об отдельном событии) и административное (предназначенное для пользователей с правами администраторов). Разработку начнем с основного представления календаря.

## Создание базы данных

В соответствии с подготовленным планом действий, первое, что мы должны сделать, разрабатывая приложение, — это создать базу данных. Запустите в своей локальной среде разработки XAMPP утилиту phpMyAdmin (`http://localhost/phpmyadmin`) и откройте вкладку SQL (указанные ниже команды также можно выполнить из PHP-сценария без привлечения phpMyAdmin). Создайте базу данных `php-jquery_example`, таблицу `events` для хранения информации о событиях и пару тестовых записей, используя приведенные ниже инструкции SQL.

```
CREATE DATABASE IF NOT EXISTS `php-jquery_example`
  DEFAULT CHARACTER SET utf8
  COLLATE utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `php-jquery_example`.`events` (
  `event_id` INT(11) NOT NULL AUTO_INCREMENT,
  `event_title` VARCHAR(80) DEFAULT NULL,
  `event_desc` TEXT,
  `event_start` TIMESTAMP NOT NULL DEFAULT '0000-00-00 00:00:00',
  `event_end` TIMESTAMP NOT NULL DEFAULT '0000-00-00 00:00:00',

  PRIMARY KEY (`event_id`),
  INDEX (`event_start`)
) ENGINE=MyISAM CHARACTER SET utf8 COLLATE utf8_unicode_ci;

INSERT INTO `php-jquery_example`.`events`
  (`event_title`, `event_desc`, `event_start`, `event_end`) VALUES
  ('New Year&#039;s Day', 'Happy New Year!',
    '2010-01-01 00:00:00', '2010-01-01 23:59:59'),
  ('Last Day of January', 'Last day of the month! Yay!',
    '2010-01-31 00:00:00', '2010-01-31 23:59:59');
```

**Примечание.** Приведенные команды относятся исключительно к СУБД MySQL. Поскольку данная книга посвящена jQuery и PHP, вдаваться в детали, касающиеся СУБД MySQL, мы не будем. Более подробную информацию относительно MySQL можно найти в книге "PHP и MySQL. Библия программиста, 2-е издание" (Диалектика, 2010 г.).

После выполнения указанных команд в столбце слева отобразится новая база данных `php-jquery_example`. Щелкните сначала на имени базы данных, чтобы отобразить таблицы, а затем — на вкладке events для просмотра созданных записей (рис. 4.3).

## Класс для подключения к базе данных

Поскольку в нашем приложении в доступе к базе данных будут нуждаться несколько классов, имеет смысл создать объект, предназначенный для открытия и хранения объекта базы данных. Назовем этот объект `DB_Connect` и поместим его в файл `class.db_connect.inc.php`, который сохраним в папке `class` (`/sys/class/class.db_connect.inc.php`).

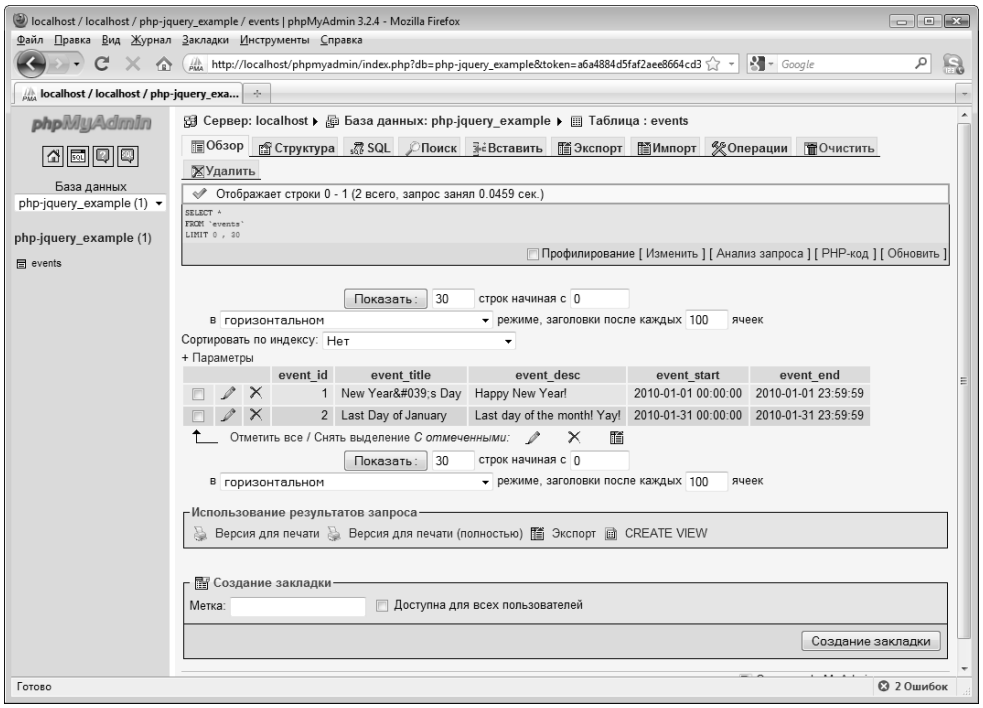



**Рис. 4.3.** Созданные база данных, таблица и записи

У этого класса будет одно свойство и один метод, оба защищенные. Назначим свойству имя `$db` и используем его для хранения объекта базы данных. Упомянутый метод — конструктор. Этот метод будет принимать в качестве необязательного параметра объект базы данных, сохраняемый далее в переменной `$db`, а если этот параметр ему не передается — создавать новый объект PDO (PHP Data Object).

Поместите в файл `class.db_connect.inc.php` приведенный ниже код.

```
<?php

/**
 * Осуществляет подключение к базе данных (доступ к БД,
 * проверка и т.п.)
 *
 * Версия PHP 5
 *
 * ЛИЦЕНЗИЯ: на этот файл распространяется лицензия MIT,
 * доступная по адресу:
 * http://www.opensource.org/licenses/mit-license.html
 *
 * @author     Jason Lengstorf <jason.lengstorf@ennuidesign.com>
 * @copyright  2009 Ennui Design
 * @license    http://www.opensource.org/licenses/mit-license.html
 */
class DB_Connect {

    /**
```

```
     * Переменная для хранения объекта базы данных
     *
     * @var object: объект базы данных
     */
    protected $db;

     /**
     * Проверить наличие объекта БД, а в случае его отсутствия --
     * создать новый
     *
     * @param object $dbo: объект базы данных
     */
    protected function __construct($db=NULL)
    {
        if ( is_object($db) )
        {
            $this->db = $db;
        }
        else
        {
            //   Константы определены в файле
            //   /sys/config/db-cred.inc.php
            $dsn = "mysql:host=" . DB_HOST . ";dbname=" . DB_NAME;
            try
            {
                $this->db = new PDO($dsn, DB_USER, DB_PASS);
            }
            catch ( Exception $e )
            {
                // Если не удается установить соединение с БД,
                // вывести сообщение об ошибке
                die ( $e->getMessage() );
            }
        }
    }
}

?>
```

**Примечание.** В этом коде используются константы, которые на данный момент еще не определены. Файлы с определениями указанных констант будут созданы в следующем разделе.

## Создание класса-оболочки для приложения

Построение самого приложения начнем с создания файла `class.calendar.inc.php`, который поместим в папку с ограниченным доступом `sys` (`/sys/class/class.calendar.inc.php`). Чтобы иметь доступ к объекту базы данных, этот класс будет наследовать класс `DB_Connect`. Откройте файл в любом привычном для вас текстовом редакторе и создайте класс `Calendar`, содержащий следующий код.

```
<?php

/**
```

```
 * Обеспечивает создание календаря и манипулирование событиями
 *
 * Версия PHP 5
 *
 * ЛИЦЕНЗИЯ: на этот файл распространяется лицензия MIT,
 * доступная по адресу:
 * http://www.opensource.org/licenses/mit-license.html
 *
 * @author     Jason Lengstorf <jason.lengstorf@ennuidesign.com>
 * @copyright  2009 Ennui Design
 * @license    http://www.opensource.org/licenses/mit-license.html
 */
class Calendar extends DB_Connect
{
    // Сюда будут помещены свойства и методы класса
}

?>
```

После того как класс создан, в него можно включить нужные свойства и методы.

## Добавление свойств класса

Поскольку класс `Calendar` не нуждается ни в каких общедоступных свойствах, а в примерах, содержащихся в данной книге, от него не будут наследоваться никакие другие классы, все его свойства будут закрытыми.

Создадим для класса `Calendar` свойства в соответствии с намеченным ранее планом.

```
<?php

class Calendar extends DB_Connect
{

    /**
     * Дата, на основании которой должен строиться календарь
     *
     * Формат хранения: ГГГГ-ММ-ДД ЧЧ:ММ:СС
     *
     * @var string: дата, выбранная для построения календаря
     */
    private $_useDate;

    /**
     * Месяц, для которого строится календарь
     *
     * @var int: выбранный месяц
     */
    private $_m;

    /**
     * Год, из которого выбирается начальный день месяца
     *
     * @var int: выбранный год
     */
```

```
    private $_y;

    /**
     * Количество дней в выбранном месяце
     *
     * @var int: количество дней в месяце
     */
    private $_daysInMonth;

    /**
     * Индекс дня недели, с которого начинается месяц (0-6)
     *
     * @var int: день недели, с которого начинается месяц
     */
    private $_startDay;

    // Сюда будут помещены методы
}

?>
```

**Примечание.** В повторяющихся фрагментах кода Doc-блоки для краткости будут опускаться.

Перечень свойств, которые мы только что создали в соответствии с исходным планом разработки, приводится ниже.

- **$_useDate.** Данные в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС, которые следует использовать при создании календаря.
- **$_useDate.** Месяц, который следует использовать при создании календаря.
- **$_useDate.** Год, который следует использовать при создании календаря.
- **$_useDate.** Количество дней в текущем месяце.
- **$_useDate.** Индекс в интервале от 0 до 6, указывающий день недели, с которого начинается месяц.

## Создание конструктора

Теперь можем приступить к созданию конструктора. Начнем с его объявления.

```
<?php

class Calendar extends DB_Connect
{


    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;
```

```
    /**
     * Создает объект базы данных и хранит соответствующие данные
     *
     * При создании экземпляра этого класса конструктор принимает
     * объект базы данных в качестве параметра. Если значение этого
     * параметра отлично от null, оно сохраняется в закрытом
     * свойстве $_db. Если же это значение равно null, то вместо
     * этого создается и сохраняется новый PDO-объектю.
     *
     * Этот метод осуществляет сбор и сохранение дополнительной
     * информации, включая месяц, начиная с которого должен строиться
     * календарь, количество дней в указанном месяце, начальный день
     * недели этого месяца, а также текущий день недели.
     *
     * @param object $dbo: объект базы данных
     * @param string $useDate: дата, выбранная для построения календаря
     * @return void
     */
    public function __construct($dbo=NULL, $useDate=NULL)
    {

    }

}

?>
```

Конструктор принимает два необязательных параметра: первый — объект базы данных, второй — данные, которые следует использовать при отображении календаря.

### Проверка соединения с базой данных

Чтобы класс мог выполнять свои функции, ему требуется подключение к базе данных. Конструктор будет либо вызывать родительский конструктор из объекта `DB_Connect` для проверки существования объекта базы данных и его использования, если этот объект доступен, либо создавать новый объект, если конструктору этот объект не предоставляется.

Организуем указанную проверку с помощью кода, выделенного ниже полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{


    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;
```

```
    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL)
    {
        /*
         * Вызвать конструктор родительского класса для проверки
         * существования объекта базы данных
         */
        parent::__construct($dbo);
    }
}

?>
```

### Создание файла для хранения учетных данных пользователя БД

Чтобы регистрационные записи пользователей базы данных, в интересах упрощения обслуживания, хранились отдельно от остальной части приложения, мы используем конфигурационный файл. Создайте новый файл `db_cred.inc.php` в папке `config` (`/sys/config/db-cred.inc.php`). Поместите в этот файл приведенный ниже код, который создает массив `$C` (для констант) и сохраняет каждую порцию данных в виде новой пары *ключ-значение*.

```
<?php

/*
 * Создать пустой массив для хранения констант
 */
$C = array();

/*
 * URL-адрес хоста базы данных
 */
$C['DB_HOST'] = 'localhost';

/*
 * Имя пользователя базы данных
 */
$C['DB_USER'] = 'root';

/*
 * Пароль доступа к базе данных
 */
$C['DB_PASS'] = '';

/*
 * Имя открываемой базы данных
 */
$C['DB_NAME'] = 'php-jquery_example';

?>
```

**Примечание.** Инициализация объекта `$C` в виде пустого массива — это мера предосторожности, исключающая возможное влияние любого "мусора", который может находиться в ячейках памяти, занимаемых объектом `$C` и определяемых как константы. Старайтесь всегда придерживаться подобной практики, особенно если речь идет о критически важных данных.

Сохраните файл. Если вы не пользуетесь XAMPP или используете для доступа к базе данных учетную запись пользователя, отличную от той, которая задана по умолчанию, подставьте в код свои значения имени хоста, имени пользователя, пароля и имени базы данных.

### Создание файла инициализации

Учетные данные пользователя БД пока что не сохранены как константы. Для решения этой задачи мы используем файл инициализации.

**Файлы инициализации** осуществляют сбор данных, загружают файлы и организуют информацию для использования в приложении. В нашем примере этот файл будет отвечать за загрузку и определение всех необходимых констант, создание объекта базы данных и настройку функции автозагрузки для классов.

Создайте файл с именем `init.inc.php` в папке `core` (`/sys/core/init.inc.php`) и введите в него следующий код.

```
<?php

/*
 * Включить необходимую конфигурационную информацию
 */
include_once '../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать PDO-объект
 */
$dsn = "mysql:host=" . DB_HOST . ";dbname=" . DB_NAME;
$dbo = new PDO($dsn, DB_USER, DB_PASS);

/*
 * Определить для классов функцию автозагрузки
 */
function __autoload($class)
{
    $filename = "../sys/class/class." . $class . ".inc.php";
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}

?>
```

Функция автозагрузки вызывается в тех случаях, когда в сценарии делается попытка создания экземпляра класса, но сам класс к этому времени еще не был загружен. Это весьма полезное средство, позволяющее легко осуществлять загрузку файлов в сценарий по требованию. Более подробную информацию относительно функции автоматической загрузки можно найти по адресу `http://php.net/autoload`.

### Создание индексного файла для объединения всех частей приложения

Чтобы увидеть, как это все работает, измените содержимое файла `index.php`, находящегося в папке `public`. Для этого достаточно включить в него файл инициализации и создать экземпляр класса `Calendar`. Наконец, убедитесь в том, что класс нормально загрузился, и, если это так, выведите структуру объекта.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь для января
 */
$cal = new Calendar($dbo, "2010-01-01 12:00:00");

if ( is_object ($cal) )
{
    echo "<pre>", var_dump($cal), "</pre>";
}

?>
```

Перейдя в браузере на сайт `http://localhost/`, вы должны увидеть в окне браузера следующее сообщение.

```
object(Calendar)#2 (6) {
  ["_useDate":"Calendar":private]=>
  NULL
  ["_m":"Calendar":private]=>
  NULL
  ["_y":"Calendar":private]=>
  NULL
  ["_daysInMonth":"Calendar":private]=>
  NULL
  ["_startDay":"Calendar":private]=>
  NULL
  ["db":protected]=>
  object(PDO)#1 (0) {
  }
}
```

### Установка базовых свойств

Создав необходимую инфраструктуру в целом, можно вернуться к конструктору класса `Calendar` и заняться его окончательной доработкой.

Проверив наличие объекта базы данных, конструктор должен сохранить определенную информацию о месяце, который будет использоваться при построении календаря.

Прежде всего конструктор проверяет, передана ли ему дата; если это так, то он сохраняет ее в свойстве `$_useDate`, в противном случае использует текущую дату.

После этого дата преобразуется в отметки времени UNIX (исчисляются в секундах от начала "эры UNIX"[1]; более подробно об этом можно прочитать в статье Википедии по адресу http://ru.wikipedia.org/wiki/UNIX-время), на основании которых рассчитываются значения года и месяца, которые сохраняются соответственно в переменных `$_m` и `$_y`.

Наконец, значения, сохраненные в переменных `$_m` и `$_y`, используются для того, чтобы определить, сколько дней содержит используемый месяц и с какого дня недели он начинается.

Соответствующую функциональность в конструкторе обеспечивает фрагмент кода, выделенный ниже полужирным шрифтом:

```
<?php

class Calendar extends DB_Connect
{


    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL)
    {
        /*
         * Вызвать конструктор родительского класса для проверки
         * существования объекта базы данных
         */
        parent::__construct($dbo);

        /*
         * Собрать и сохранить информацию, относящуюся к месяцу
         */
        if ( isset($useDate) )
        {
             $this->_useDate = $useDate;
        }
        else
        {
             $this->_useDate = date('Y-m-d H:i:s');
        }

        /*
         * Преобразовать во временную метку UNIX, а затем
         * определить месяц и год, которые следует использовать
         * при построении календаря
```

[1] Моментом начала "эры UNIX" считается полночь (по UTC) с 31 декабря 1969 года на 1 января 1970 года. — *Примеч. ред.*

```
         */
        $ts = strtotime($this->_useDate);
        $this->_m = date('m', $ts);
        $this->_y = date('Y', $ts);

        /*
         * Определить количество дней, содержащихся в месяце
         */
        $this->_daysInMonth = cal_days_in_month(
                CAL_GREGORIAN,
                $this->_m,
                $this->_y
            );

        /*
         * Определить, с какого дня недели начинается месяц
         */
        $ts = mktime(0, 0, 0, $this->_m, 1, $this->_y);
        $this->_startDay = date('w', $ts);
    }
}

?>
```

При перезагрузке страницы `http://localhost/` все свойства, значения которых ранее были равны `NULL`, получат нужные значения.

```
object(Calendar)#2 (6) {
  ["_useDate":"Calendar":private]=>
  string(19) "2010-01-01 12:00:00"
  ["_m":"Calendar":private]=>
  string(2) "01"
  ["_y":"Calendar":private]=>
  string(4) "2010"
  ["_daysInMonth":"Calendar":private]=>
  int(31)
  ["_startDay":"Calendar":private]=>
  string(1) "5"
  ["db":protected]=>
  object(PDO)#1 (0) {
  }
}
```

## Загрузка информации о событиях

Для загрузки данных, описывающих события, потребуется создать новый метод, который будет обращаться к базе данных и извлекать из нее необходимую информацию. Поскольку доступ к данным будет осуществляться двумя способами (о втором из них речь пойдет немного позже), мы постараемся сделать этот метод достаточно универсальным, чтобы его можно было использовать в обоих случаях.

Назовем этот метод `_loadEventData()` и объявим его закрытым. Метод принимает один необязательный параметр — уникальный идентификатор события (ID) — и загружает события, выполняя действия в соответствии со следующей процедурой.

1. Создать базовый запрос SELECT для загрузки доступных полей из таблицы events.
2. Проверить, не был ли передан идентификатор события, и если был, то включить в запрос предложение WHERE, чтобы ответ содержал информацию только об одном событии.
3. В противном случае выполнить оба описанных ниже действия:
   - найти моменты времени, соответствующие полуночи первого дня месяца и 11:59:59 после полудня (PM) последнего дня месяца;
   - добавить предложение WHERE...BETWEEN для загрузки информации, относящейся лишь к тем датам, которые попадают в интервал, определяемый текущим месяцем.
4. Выполнить запрос.
5. Возвратить ассоциативный массив результатов.

Реализованный с учетом этих положений метод выделен в приведенном ниже коде полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{


    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    /**
     * Загружает информацию о событии (событиях) в массив
     *
     * @param int $id: необязательный идентификатор (ID),
     * используемый для фильтрации результатов
     * @return array: массив событий, извлеченных из базы данных
     */
    private function _loadEventData($id=NULL)
    {
        $sql = "SELECT
                    `event_id`, `event_title`, `event_desc`,
                    `event_start`, `event_end`
                FROM `events`";
        /*
         * Если предоставлен идентификатор (ID) события, добавить
         * предложение WHERE, чтобы было возвращено только это событие
         */
```

```
        if ( !empty($id) )
        {
            $sql .= "WHERE `event_id`=:id LIMIT 1";
        }

        /*
         * В противном случае загрузить все события, относящиеся к
         * используемому месяцу
         */
        else
        {
            /*
             * Найти первый и последний дни месяца
             */
            $start_ts = mktime(0, 0, 0, $this->_m, 1, $this->_y);
            $end_ts = mktime(23, 59, 59, $this->_m+1, 0, $this->_y);
            $start_date = date('Y-m-d H:i:s', $start_ts);
            $end_date = date('Y-m-d H:i:s', $end_ts);

            /*
             * Отфильтровать события, выбрав из них только те,
             * которые относятся к месяцу, выбранному текущим
             */
            $sql .= "WHERE `event_start`
                        BETWEEN '$start_date'
                        AND '$end_date'
                    ORDER BY `event_start`";
        }

        try
        {
            $stmt = $this->db->prepare($sql);

            /*
             * Привязать параметр, если был передан идентификатор
             */
            if ( !empty($id) )
            {
                $stmt->bindParam(":id", $id, PDO::PARAM_INT);
            }

            $stmt->execute();
            $results = $stmt->fetchAll(PDO::FETCH_ASSOC);
            $stmt->closeCursor();

            return $results;
        }
        catch ( Exception $e )
        {
            die ( $e->getMessage() );
        }
    }
}

?>
```

**Примечание.** Методы, ссылки на которые отсутствуют, для краткости отображены в свернутом виде.

Данный метод возвращает массив, который в случае использования тестовых записей, созданных нами ранее в базе данных, состоит из следующих элементов.

```
array(2) {
  [0]=>
  array(5) {
    ["event_id"]=>
    string(1) "1"
    ["event_title"]=>
    string(19) "New Year's Day"
    ["event_desc"]=>
    string(15) "Happy New Year!"
    ["event_start"]=>
    string(19) "2010-01-01 00:00:00"
    ["event_end"]=>
    string(19) "2010-01-01 23:59:59"
  }
  [1]=>
  array(5) {
    ["event_id"]=>
    string(1) "2"
    ["event_title"]=>
    string(19) "Last Day of January"
    ["event_desc"]=>
    string(27) "Last day of the month! Yay!"
    ["event_start"]=>
    string(19) "2010-01-31 00:00:00"
    ["event_end"]=>
    string(19) "2010-01-31 23:59:59"
  }
}
```

### *Создание массива объектов событий для использования в календаре*

В необработанном виде выходная информация метода `_loadEventData()` не совсем пригодна для непосредственного использования. Поскольку каждое событие должно отображаться в соответствующей ячейке календаря, то после получения массива событий из метода `_loadEventData` их следует сгруппировать по дням, на которые они приходятся. Кроме того, чтобы упростить работу с событиями, мы будем использовать для них упрощенные поля.

Нашей конечной целью является создание такого массива событий, в котором порядковый номер дня месяца служит индексом, а каждое событие является индексированным объектом. По завершении разработки нового метода тестовые записи в нашей базе данных должны храниться в следующем виде.

```
array(2) {
  [1]=>
  array(1) {
    [0]=>
    object(Event)#3 (5) {
      ["id"]=>
      string(1) "1"
      ["title"]=>
```

```
      string(19) "New Year's Day"
      ["description"]=>
      string(15) "Happy New Year"
      ["start"]=>
      string(19) "2010-01-01 00:00:00"
      ["end"]=>
      string(19) "2010-01-01 23:59:59"
    }
  }
  [31]=>
  array(1) {
    [0]=>
    object(Event)#4 (5) {
      ["id"]=>
      string(1) "2"
      ["title"]=>
      string(19) "Last Day of January"
      ["description"]=>
      string(27) "Last day of the month! Yay!"
      ["start"]=>
      string(19) "2010-01-31 00:00:00"
      ["end"]=>
      string(19) "2010-01-31 23:59:59"
    }
  }
}
```

### Создание класса события

Для достижения намеченной нами цели необходимо сначала создать новый класс, который мы назовем `Event` и поместим в папку `class` (`/sys/class/class.event.inc.php`). У класса будет пять общедоступных свойств: `$id`, `$title`, `$description`, `$start` и `$end`, а также конструктор, который устанавливает каждое из этих свойств с помощью ассоциативного массива, возвращаемого запросом к базе данных. Создайте этот файл и введите в него следующий код.

```
<?php

/**
 * Хранит информацию о событии
 *
 * Версия PHP 5
 *
 * ЛИЦЕНЗИЯ: на этот файл распространяется лицензия MIT,
 * доступная по адресу:
 * http://www.opensource.org/licenses/mit-license.html
 *
 * @author     Jason Lengstorf <jason.lengstorf@ennuidesign.com>
 * @copyright  2009 Ennui Design
 * @license    http://www.opensource.org/licenses/mit-license.html
 */
class Event
{
```

```
/**
 * Идентификатор (ID) события
 *
 * @var int
 */
public $id;

/**
 * Название события
 *
 * @var string
 */
public $title;

/**
 * Описание события
 *
 * @var string
 */
public $description;

/**
 * Время начала события
 *
 * @var string
 */
public $start;

/**
 * Время окончания события
 *
 * @var string
 */
public $end;

/**
 * Принимает массив данных о событии и сохраняет его
 *
 * @param array $event: ассоциативный массив данных о событии
 * @return void
 */
public function __construct($event)
{
    if ( is_array($event) )
    {
        $this->id = $event['event_id'];
        $this->title = $event['event_title'];
        $this->description = $event['event_desc'];
        $this->start = $event['event_start'];
        $this->end = $event['event_end'];
    }
    else
    {
        throw new Exception("Не были предоставлены данные о событии.");
```

```
        }
    }

}
?>
```

### Создание метода, сохраняющего объекты события в виде массива

Теперь, когда у нас имеется возможность хранить каждое событие в виде объекта, можно создать метод, который перебирает все имеющиеся события и сохраняет их в виде массива в соответствии с датами событий. Прежде всего загрузим дату события из базы данных с помощью метода `_loadEventData()`. Затем извлечем день месяца из начальной даты каждого события и добавим новое событие в элемент массива, индекс которого определяется этим днем. Для этого создадим в классе `Calendar` новый закрытый метод, который назовем `_createEventObj()`. Загрузка события из базы данных и создание нового массива осуществляются с помощью кода, выделенного ниже полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    private function _loadEventData($id=NULL) {...}

/**
 * Загружает все события, относящиеся к данному месяцу, в массив
 *
 * @return array: информация о событиях
 */
private function _createEventObj()
{
    /*
     * Загрузить массив событий
     */
    $arr = $this->_loadEventData();

    /*
     * Создать новый массив, а затем организовать события в
     * зависимости от дня месяца, в который они происходят
     */
    $events = array();
```

```
    foreach ( $arr as $event )
    {
        $day = date('j', strtotime($event['event_start']));
        try
        {
            $events[$day][] = new Event($event);
        }
        catch ( Exception $e )
        {
            die ( $e->getMessage() );
        }
    }
    return $events;
}

}
?>
```

Теперь, когда события могут быть загружены и организованы так, чтобы метод отображал календарь в привычном виде, можно без труда разместить даты в нужных ячейках с помощью HTML.

## Вывод HTML-кода для отображения календаря и событий

К этому моменту мы располагаем настроенной базой данных, сохраненными тестовыми событиями и методами, позволяющими загрузить даты событий и организовать их в виде легкодоступного массива. Теперь можно свести все воедино и закончить построение календаря.

Присвоим используемому для этих целей методу имя `buildCalendar()`. Метод будет генерировать календарь, имеющий следующие атрибуты:

- заголовок, который будет отображать используемый месяц и год;
- сокращенные обозначения дней недели, необходимые для придания календарю присущего ему вида;
- пронумерованные ячейки, содержащие события, если они существуют для данной даты.

Прежде всего объявим метод `buildCalendar()` в классе `Calendar` и создадим заголовок в элементе `H2`. Кроме того, создадим массив сокращенных обозначений дней недели и организуем по ним цикл, генерирующий неупорядоченный список. Для этого используем дополнительный код, выделенный в приведенном ниже листинге полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{


    private $_useDate;

    private $_m;

    private $_y;
```

```
    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    /**
     * Возвращает HTML-разметку для отображения календаря и событий
     *
     * На основании информации, хранящейся в свойствах класса,
     * загружаются события для данного месяца, генерируется
     * календарь и возвращается актуальная разметка.
     *
     * @return string: HTML-разметка календаря
     */
    public function buildCalendar()
    {
        /*
         * Определить месяц календаря и создать массив сокращенных
         * обозначений дней недели, которые будут использованы
         * в заголовках столбцов
         */
        $cal_month = date('F Y', strtotime($this->_useDate));
        $weekdays = array('Sun', 'Mon', 'Tue', 'Wed', 'Thu',
                          'Fri', 'Sat');
        /*
         * Добавить заголовок в HTML-разметку календаря
         */
        $html = "\n\t<h2>$cal_month</h2>";
        for ( $d=0, $labels=NULL; $d<7; ++$d )
        {
            $labels .= "\n\t\t<li>" . $weekdays[$d] . "</li>";
        }
        $html .= "\n\t<ul class=\"weekdays\">"
            . $labels . "\n\t</ul>";
        /*
         * Возвратить HTML-разметку для вывода
         */
        return $html;
    }
}

?>
```

### Модифицированный файл `index.php`

Чтобы просмотреть вывод метода `buildCalendar`, необходимо модифицировать файл `index.php`, хранящийся в папке `public`, организовав в нем вызов этого метода. Обновите файл, добавив в него код, выделенный в приведенном ниже листинге полужирным шрифтом.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь для января
 */
$cal = new Calendar($dbo, "2010-01-01 12:00:00");

/*
 * Отобразить календарь в виде HTML
 */
echo $cal->buildCalendar();

?>
```

А теперь взгляните, чего мы достигли на данном этапе (рис. 4.4).

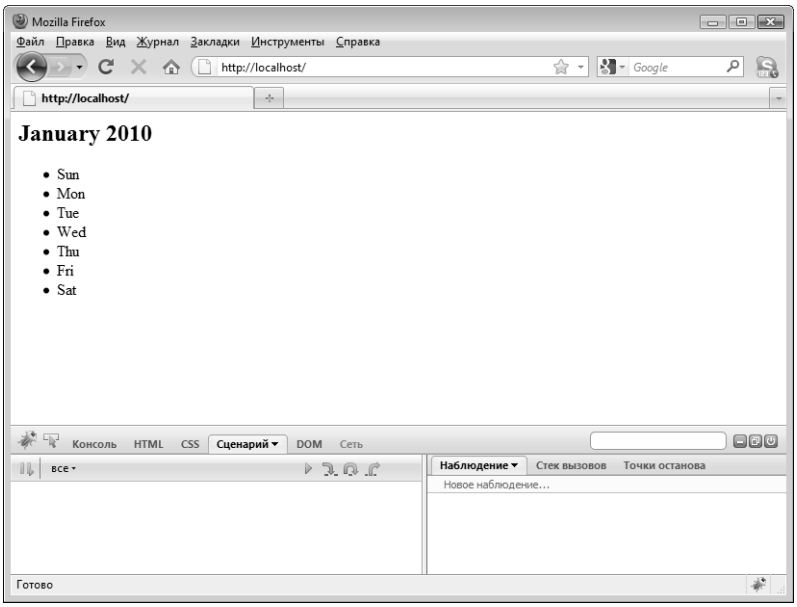



**Рис. 4.4.** Вывод заголовка календаря и сокращенных обозначений дней недели

## *Построение календаря*

Наша следующая задача — построение фактической раскладки календарных дней. Для ее решения потребуется выполнить ряд действий, перечень которых приводится ниже.

1. Создать новый неупорядоченный список.
2. Организовать цикл (со счетчиком итераций, счетчиком календарных дат и текущими значениями даты, месяца и года, хранящимися в переменных),

который выполняется до тех пор, пока значение счетчика календарных дат остается меньшим количества дней в месяце.

3. Добавить класс `fill` в дни недели, предшествующие первому.
4. Добавить класс `today`, если текущая дата принадлежит тому же месяцу и году, что и генерируемая дата, и совпадает с ней.
5. Создать открывающий и закрывающий дескрипторы элемента списка для каждого дня.
6. Проверить, приходится ли текущая ячейка календаря на текущий месяц, и если это так, то добавить дату.
7. Проверить, является ли текущая ячейка календаря субботой, и если это так, то закрыть список и открыть новый.
8. Собрать воедино отдельные части элемента списка и присоединить их к разметке.
9. По завершении этого цикла запустить другой цикл, добавляющий заполнители дней до конца календарной недели.
10. Закрыть окончательный неупорядоченный список и возвратить разметку.

Начнем с выполнения пп. 1 и 2, добавив в метод `buildCalendar()` код, выделенный в приведенном ниже листинге полужирным шрифтом.

```
public function buildCalendar()
{
    /*
     * Определить месяц календаря и создать массив сокращенных
     * обозначений дней недели, которые будут использованы
     * в заголовках столбцов
     */
    $cal_month = date('F Y', strtotime($this->_useDate));
    $weekdays = array('Sun', 'Mon', 'Tue', 'Wed', 'Thu',
                      'Fri', 'Sat');
    /*
     * Добавить заголовок в HTML-разметку календаря
     */
    $html = "\n\t<h2>$cal_month</h2>";
    for ( $d=0, $labels=NULL; $d<7; ++$d )
    {
        $labels .= "\n\t\t<li>" . $weekdays[$d] . "</li>";
    }
    $html .= "\n\t<ul class=\"weekdays\">"
        . $labels . "\n\t</ul>";

    /*
     * Создать HTML-разметку календаря
     */
    $html .= "\n\t<ul>"; // Начать новый неупорядоченный список
    for ( $i=1, $c=1, $t=date('j'), $m=date('m'), $y=date('Y');
            $c<=$this->_daysInMonth; ++$i )
    {
        // Сюда будут помещены дополнительные операции
    }

    /*
```

```
     * Возвратить HTML-разметку для вывода
     */
    return $html;
}
```

Для выполнения действий, указанных в пп. 3–5, добавьте следующий код, выделенный полужирным шрифтом.

```
public function buildCalendar()
{
    /*
     * Определить месяц календаря и создать массив сокращенных
     * обозначений дней недели, которые будут использованы
     * в заголовках столбцов
     */
    $cal_month = date('F Y', strtotime($this->_useDate));
    $weekdays = array('Sun', 'Mon', 'Tue', 'Wed', 'Thu',
                      'Fri', 'Sat');
    /*

     * Добавить заголовок в HTML-разметку календаря
     */
    $html = "\n\t<h2>$cal_month</h2>";
    for ( $d=0, $labels=NULL; $d<7; ++$d )
    {
        $labels .= "\n\t\t<li>" . $weekdays[$d] . "</li>";
    }
    $html .= "\n\t<ul class=\"weekdays\">" . $labels .
             "\n\t</ul>";

    /*
     * Создать HTML-разметку календаря
     */
    $html .= "\n\t<ul>"; // Start a new unordered list
    for ( $i=1, $c=1, $t=date('j'), $m=date('m'), $y=date('Y');
          $c<=$this->_daysInMonth; ++$i )
    {
        /*
         * Применить класс "fill" к ячейкам календаря,
         * располагающимся перед первым днем данного месяца
         */
        $class = $i<=$this->_startDay ? "fill" : NULL;

        /*
         * Добавить класс "today", если дата совпадает с текущей
         */
        if ( $c==$t && $m==$this->_m && $y==$this->_y )
        {
            $class = "today";
        }

        /*
         * Создать открывающий и закрывающий дескрипторы
         * элемента списка
         */
```

```
            $ls = sprintf("\n\t\t<li class=\"%s\">", $class);
            $le = "\n\t\t</li>";

            // Сюда будут помещены дополнительные операции
        }

        /*
         * Возвратить HTML-разметку для вывода
         */
        return $html;
    }
```

Наконец, для выполнения действий, указанных в пп. 6–10, введите еще один дополнительный фрагмент кода, как показано ниже.

```
    public function buildCalendar()
    {
        /*
         * Определить месяц календаря и создать массив сокращенных
         * обозначений дней недели, которые будут использованы
         * в заголовках столбцов
         */
        $cal_month = date('F Y', strtotime($this->_useDate));
        $weekdays = array('Sun', 'Mon', 'Tue', 'Wed', 'Thu',
                          'Fri', 'Sat');

        /*
         * Добавить заголовок в HTML-разметку календаря
         */
        $html = "\n\t<h2>$cal_month</h2>";
        for ( $d=0, $labels=NULL; $d<7; ++$d )
        {
            $labels .= "\n\t\t<li>" . $weekdays[$d] . "</li>";
        }
        $html .= "\n\t<ul class=\"weekdays\">" . $labels .
                 "\n\t</ul>";

        /*
         * Создать HTML-разметку календаря
         */
        $html .= "\n\t<ul>"; // Start a new unordered list
        for ( $i=1, $c=1, $t=date('j'), $m=date('m'), $y=date('Y');
              $c<=$this->_daysInMonth; ++$i )
        {
            /*
             * Применить класс "fill" к ячейкам календаря,
             * располагающимся перед первым днем данного месяца
             */
            $class = $i<=$this->_startDay ? "fill" : NULL;

            /*
             * Добавить класс "today", если дата совпадает с текущей
             */
            if ( $c==$t && $m==$this->_m && $y==$this->_y )
            {
```

```
            $class = "today";
        }

        /*
         * Создать открывающий и закрывающий дескрипторы
         * элемента списка
         */
        $ls = sprintf("\n\t\t<li class=\"%s\">", $class);
        $le = "\n\t\t</li>";

        /*
         * Добавить день месяца, идентифицирующий ячейку
         * календаря
         */
        if ( $this->_startDay<$i && $this->_daysInMonth>=$c)
        {
            $date = sprintf("\n\t\t\t<strong>%02d</strong>",$c++);
        }
        else { $date=" "; }

        /*
         * Если текущий день суббота, перейти в следующий ряд
         */
        $wrap = $i!=0 && $i%7==0 ? "\n\t</ul>\n\t<ul>" : NULL;

        /*
         * Собрать разрозненные части воедино
         */
        $html .= $ls . $date . $le . $wrap;
    }

    /*
     * Добавить заполнители для завершения последней недели
     */
    while ( $i%7!=1 )
    {
        $html .= "\n\t\t<li class=\"fill\"> </li>";++$i;
    }

    /*
     * Закрыть окончательный неупорядоченный список
     */
    $html .= "\n\t</ul>\n\n";

    /*
     * Возвратить HTML-разметку для вывода
     */
    return $html;
}
```

Протестировав текущую версию этой функции, вы должны увидеть в браузере неупорядоченный список (рис. 4.5).

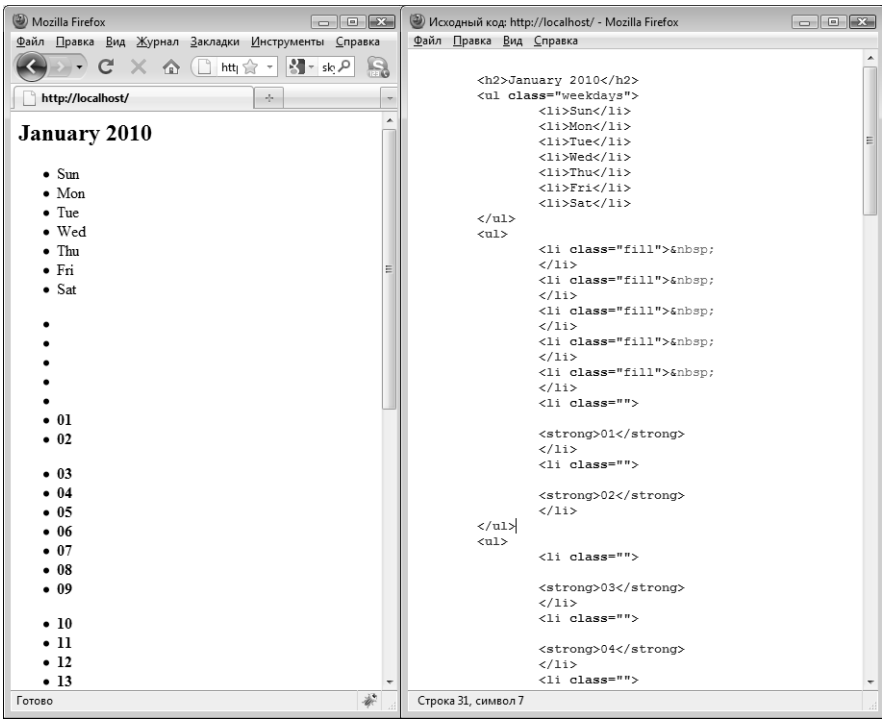



**Рис. 4.5.** Разметка, сгенерированная функцией `buildCalendar()`

## *Отображение событий в календаре*

Добавление событий в отображение календаря сводится к загрузке массива `events` из функции `_createEventObj()` и организации цикла для поиска событий с индексом, совпадающим с текущим днем, если таковые существуют. Информация о событии добавляется в разметку календаря с помощью кода, выделенного в приведенном ниже листинге полужирным шрифтом.

```
public function buildCalendar()
{
    /*
     * Определить месяц календаря и создать массив сокращенных
     * обозначений дней недели, которые будут использованы
     * в заголовках столбцов
     */
    $cal_month = date('F Y', strtotime($this->_useDate));
    $weekdays = array('Sun', 'Mon', 'Tue', 'Wed', 'Thu',
                      'Fri', 'Sat');

    /*
     * Добавить заголовок в HTML-разметку календаря
     */
    $html = "\n\t<h2>$cal_month</h2>";
    for ( $d=0, $labels=NULL; $d<7; ++$d )
    {
```

```
    $labels .= "\n\t\t<li>" . $weekdays[$d] . "</li>";
}
$html .= "\n\t<ul class=\"weekdays\">" . $labels .
         "\n\t</ul>";

/*
 * Загрузить данные о событиях
 */
$events = $this->_createEventObj();

/*
 * Создать HTML-разметку календаря
 */
$html .= "\n\t<ul>"; // Start a new unordered list
for ( $i=1, $c=1, $t=date('j'), $m=date('m'), $y=date('Y');
      $c<=$this->_daysInMonth; ++$i )
{
    /*
     * Применить класс "fill" к ячейкам календаря,
     * располагающимся перед первым днем данного месяца
     */
    $class = $i<=$this->_startDay ? "fill" : NULL;

    /*
     * Добавить класс "today", если дата совпадает с текущей
     */
    if ( $c==$t && $m==$this->_m && $y==$this->_y )
    {
        $class = "today";
    }

    /*
     * Создать открывающий и закрывающий дескрипторы
     * элемента списка
     */
    $ls = sprintf("\n\t\t<li class=\"%s\">", $class);
    $le = "\n\t\t</li>";

    /*
     * Добавить день месяца, идентифицирующий ячейку
     * календаря
     */
    if ( $this->_startDay<$i && $this->_daysInMonth>=$c)
    {
        /*
         * Форматировать данные о событиях
         */
        $event_info = NULL; // clear the variable
        if ( isset($events[$c]) )
        {
            foreach ( $events[$c] as $event )
            {
                $link = '<a href="view.php?event_id='
                        . $event->id . '">' . $event->title
                        . '</a>';
```

```
                    $event_info .= "\n\t\t\t$link";
                }
            }

            $date = sprintf("\n\t\t\t<strong>%02d</strong>",$c++);
        }
        else { $date=" "; }

        /*
         * Если текущий день суббота, перейти в следующий ряд
         */
        $wrap = $i!=0 && $i%7==0 ? "\n\t</ul>\n\t<ul>" : NULL;

        /*
         * Собрать разрозненные части воедино
         */
        $html .= $ls . $date . $event_info . $le . $wrap;
    }

    /*
     * Добавить заполнители для завершения последней недели
     */
    while ( $i%7!=1 )
    {
        $html .= "\n\t\t<li class=\"fill\"> </li>";++$i;
    }

    /*
     * Закрыть окончательный неупорядоченный список
     */
    $html .= "\n\t</ul>\n\n";

    /*
     * Возвратить HTML-разметку для вывода
     */
    return $html;
}
```

**Предупреждение.** Обратите внимание на добавление новой переменной `$event_info` в разметку в нижней части цикла.

Когда события из базы данных загружены в вывод календаря, их названия отображаются рядом с соответствующими датами (рис. 4.6).

**Примечание.** Ссылки в виде названий событий указывают на файл `view.php`, который пока еще не существует. Этот файл будет создан и описан далее.

### *Придание календарю свойственного ему вида*

На данном этапе у нас имеется правильно работающая разметка, в которую включаются события, но сгенерированный код не обеспечивает отображение календаря в том виде, к которому мы все привыкли.

Для исправления этого недостатка потребуется определенная доработка HTML-разметки, которая позволит изменить внешний вид страницы с помощью CSS.

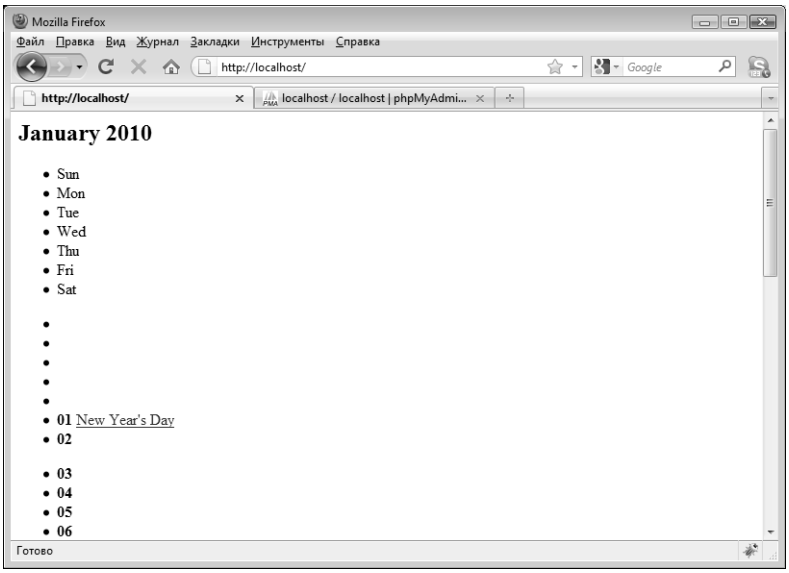



**Рис. 4.6.** Отображение названий событий для соответствующих дат

**Примечание.** Мы не будем вдаваться в детальные объяснения используемых правил стилевой разметки, поскольку CSS не является предметом рассмотрения данной книги. Для получения более подробной информации относительно CSS можно обратиться к книге "HTML, XHTML и CSS. Библия пользователя, 5-е издание" (Диалектика, 2010 г.).

В целом, используемый нами CSS-файл решает следующие задачи:

- перемещает все элементы списка влево;
- устанавливает поля и границы таким образом, чтобы даты располагались, как в обычном календаре;
- добавляет эффект "наведения", благодаря которому ячейка календаря, над которой располагается указатель мыши, выделяется подсветкой;
- выделяет стилем названия событий;
- добавляет эффект "наведения" также для названий событий;
- применяет некоторые элементы оформления из CSS3, включая закругленные углы и тени, для улучшения внешнего вида календаря.

**Совет.** Для получения более подробной информации о CSS3 посетите сайт `http://css3.info/`.

Создайте в папке `css` новый файл с именем `style.css` (`/public/assets/css/style.css`) и введите в него следующий код CSS.

```
body {
    background-color: #789;
    font-family: georgia, serif;
    font-size: 13px;
}

#content {
```

```
    display: block;
    width: 812px;
    margin: 40px auto 10px;
    padding: 10px;
    background-color: #FFF;
    -moz-border-radius: 6px;
    -webkit-border-radius: 6px;
    border-radius: 6px;
    border:2px solid black;
    -moz-box-shadow: 0 0 14px #123;
    -webkit-box-shadow: 0 0 14px #123;
    box-shadow: 0 0 14px #123;
}

h2,p {
    margin: 0 auto 14px;
    text-align: center;
}

ul {
    display: block;
    clear: left;
    height: 82px;
    width: 812px;
    margin: 0 auto;
    padding: 0;
    list-style: none;
    background-color: #FFF;
    text-align: center;
    border: 1px solid black;
    border-top: 0;
    border-bottom: 2px solid black;
}

li {
    position: relative;
    float: left;
    margin: 0;
    padding: 20px 2px 2px;
    border-left: 1px solid black;
    border-right: 1px solid black;
    width: 110px;
    height: 60px;
    overflow: hidden;
    background-color: white;
}

li:hover {
    background-color: #FCB;
    z-index: 1;
    -moz-box-shadow: 0 0 10px #789;
    -webkit-box-shadow: 0 0 10px #789;
    box-shadow: 0 0 10px #789;
}
```

```
.weekdays {
    height: 20px;
    border-top: 2px solid black;
}

.weekdays li {
    height: 16px;
    padding: 2px 2px;
    background-color: #BCF;
}

.fill {
    background-color: #BCD;
}

.weekdays li:hover,li.fill:hover {
    background-color: #BCD;
    -moz-box-shadow: none;
    -webkit-box-shadow: none;
    box-shadow: none;
}

.weekdays li:hover,.today {
    background-color: #BCF;
}

li strong {
    position: absolute;
    top: 2px;
    right: 2px;
}

li a {
    position: relative;
    display: block;
    border: 1px dotted black;
    margin: 2px;
    padding: 2px;
    font-size: 11px;
    background-color: #DEF;
    text-align: left;
    -moz-border-radius: 6px;
    -webkit-border-radius: 6px;
    border-radius: 6px;
    z-index: 1;
    text-decoration: none;
    color: black;
    font-weight: bold;
    font-style: italic;
}

li a:hover {
    background-color: #BCF;
    z-index: 2;
```

```
    -moz-box-shadow: 0 0 6px #789;
    -webkit-box-shadow: 0 0 6px #789;
    box-shadow: 0 0 6px #789;
}
```

Сохраните таблицу стилей и закройте ее; в этой главе она не будет претерпевать изменений. В следующем разделе будут созданы общие для всех страниц файлы, которые, кроме всего прочего, будут включать эти стили в страницы.

### Создание общих файлов — верхнего и нижнего колонтитулов страницы

Пользователи вашего приложения будут просматривать множество страниц, относящихся к различным датам, но все страницы должны иметь одинаковый внешний вид, и, следовательно, должен быть предусмотрен общий для всех страниц набор HTML-элементов, таблиц стилей и т.п. Чтобы упростить обслуживание приложения, общие элементы будут храниться в двух файлах: `header.inc.php` (верхний колонтитул) и `footer.inc.php` (нижний колонтитул).

Прежде всего создайте файл `header.inc.php`, поместив его в папку `common` (`/public/assets/common/header.inc.php`). В начале файла будут располагаться объявление `DOCTYPE` для HTML и раздел заголовка, содержащий дескриптор метаданных `Content-Type`, название документа и ссылки на используемые CSS-файлы.

При переходе от одной страницы к другой название документа будет меняться, поэтому для хранения текущего названия используется переменная `$page_title`.

Кроме того, поскольку страница может нуждаться в нескольких CSS-файлах, предусмотрена передача массива имен файлов в переменной `$css_files` и их циклический просмотр для генерации соответствующей разметки.

Введите в файл `header.inc.php` следующий код.

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
          content="text/html;charset=utf-8" />
    <title><?php echo $page_title; ?></title>
<?php foreach ( $css_files as $css ): ?>
    <link rel="stylesheet" type="text/css" media="screen,projection"
          href="assets/css/<?php echo $css; ?>" />
<?php endforeach; ?>
</head>

<body>
```

После этого создайте в папке `common` файл `footer.inc.php` (`/public/assets/common/footer.inc.php`), содержащий закрывающие дескрипторы разметки.

Пока что данный файл будет всего лишь закрывать дескрипторы `body` и `html`, открытые в файле `header.inc.php`, однако по мере дальнейшей разработки приложения его содержимое будет пополняться.

Введите в файл `footer.inc.php` следующий код.

```
</body>

</html>
```

### Включение файлов в файл index.php

Чтобы новые фрагменты кода можно было использовать, потребуется внести некоторые изменения в файл index.php. Прежде всего включите в него файл header.inc.php и присвойте значения переменным $page_title и $css_files.

Кроме того, создайте контейнер для содержимого страницы, добавив новый элемент div с идентификатором content, служащий оболочкой для вызова метода buildCalendar().

Наконец, добавьте вызов файла footer.inc.php для завершения страницы. Когда вы все это сделаете, файл index.php будет иметь следующий вид, где изменения выделены полужирным шрифтом.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь для января
 */
$cal = new Calendar($dbo, "2010-01-01 12:00:00");

/*
 * Задать название страницы и файлы CSS
 */
$page_title = "Календарь событий";
$css_files = array('style.css');

/*
 * Включить начальную часть страницы
 */
include_once 'assets/common/header.inc.php';

?>

<div id="content">
<?php

/*
 * Отобразить календарь в виде HTML
 */
echo $cal->buildCalendar();

?>

</div><!-- end #content -->
<?php

/*
 * Включить завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохраните изменения и перезагрузите браузер, чтобы посмотреть на полученный результат (рис. 4.7).

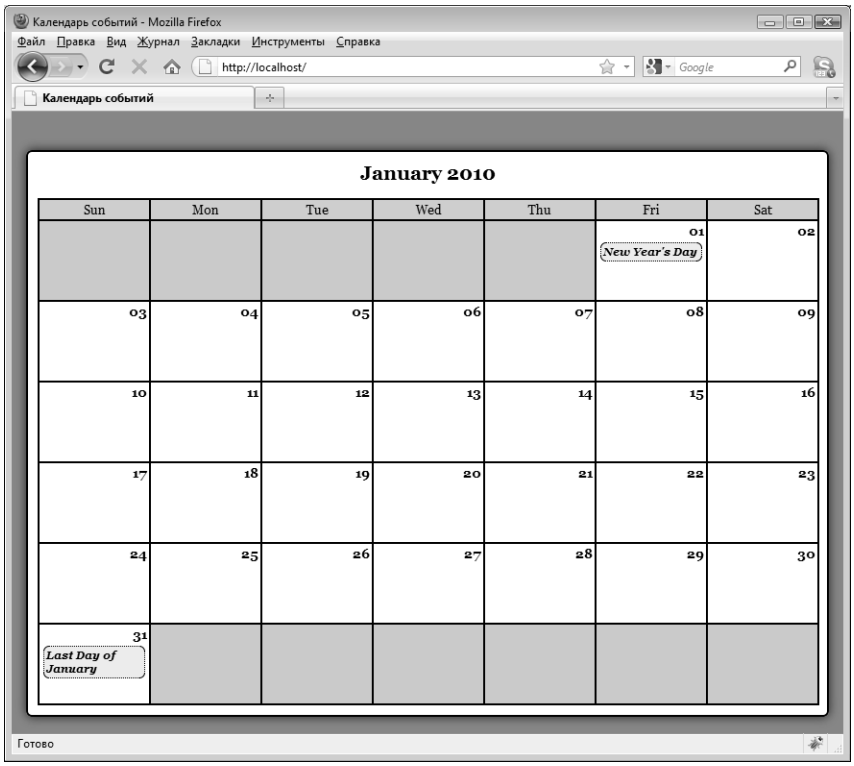



**Рис. 4.7.** Вид календаря после добавления файлов, отображающих начальную и завершающую части страницы с использованием CSS-стилей

## Вывод HTML-кода для отображения подробного описания события

Следующее, что мы должны сделать в приложении, — это предоставить пользователю возможность просматривать подробные описания событий. Решение этой задачи будет включать в себя следующие три этапа.

1. Создать метод, осуществляющий форматирование массива данных одиночных событий после их загрузки на основе идентификатора.
2. Создать метод для генерации разметки, содержащей данные, загруженные первым методом.
3. Создать новый файл для отображения разметки, сгенерированной вторым методом.

### Создание метода для форматирования данных одиночного события

Метод `_LoadEventByID()`, аналогичный по своим функциям методу `_createEventObj()`, предназначен для генерации объекта `Event` на основании результирующего набора, возвращенного методом `_loadEventData()`.

Поскольку генерация разметки при использовании только одного события осуществляется довольно просто, все, что будет делать этот метод, сводится к загрузке требуемого события по его идентификатору (ID) с помощью метода `_loadEventData()` и последующему возврату первого (и единственного, благодаря ограничению `LIMIT 1` в методе `_loadEventData()`) из полученных результатов.

Добавьте в класс `Calendar` метод, выделенный в листинге полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    public function buildCalendar() {...}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    /**
     * Возвращает объект одиночного события
     *
     * @param int $id: идентификатор (ID) события
     * @return object: объект события
     */
    private function _loadEventById($id)
    {
        /*
         * Если ID не передан, возвратить NULL
         */
        if ( empty($id) )
        {
            return NULL;
        }

        /*
         * Загрузить события в массив
         */
```

```
        $event = $this->_loadEventData($id);

        /*
         * Возвратить объект события
         */
        if ( isset($event[0]) )
        {
            return new Event($event[0]);
        }
        else
        {
            return NULL;
        }
    }

}

?>
```

В результате выполнения этого метода возвращается объект, который (для значения ID, равного 1) выглядит примерно так.

```
Event Object
(
    [id] => 1
    [title] => New Year's Day
    [description] => Happy New Year!
    [start] => 2010-01-01 00:00:00
    [end] => 2010-01-01 23:59:59
```

### Создание метода для генерации разметки

Теперь, когда у нас имеется массив данных одиночных событий, можно создать новый общедоступный метод, который будет форматировать данные, преобразуя их в HTML-разметку.

Мы назовем этот метод `displayEvent()`. Он принимает уникальный идентификатор события в качестве параметра и генерирует HTML-разметку, выполняя следующие действия:

1) загружает данные событий с помощью метода `_loadEventById()`;
2) использует начальную и конечную даты для генерации строк, описывающих событие;
3) возвращает HTML-разметку для отображения события.

Создайте метод `displayEvent()`, добавив в класс `Calendar` код, выделенный в листинге полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;
```

```
private $_y;

private $_daysInMonth;

private $_startDay;

public function __construct($dbo=NULL, $useDate=NULL) {...}

public function buildCalendar() {...}

/**
 * Отображает информацию о заданном событии
 *
 * @param int $id: идентификатор (ID) события
 * @return string: элементарная разметка для отображения
 *                 информации о событии
 */
public function displayEvent($id)
{
/*
 * Убедиться в том, что ID был передан
 */
if ( empty($id) ) { return NULL; }

/*
 * Проверить, что ID является целым числом
 */
$id = preg_replace('/[^0-9]/', '', $id);

/*
 * Загрузить данные о событии из БД
 */
$event = $this->_loadEventById($id);

/*
 * Сгенерировать строки для даты, начального и конечного времени
 */
$ts = strtotime($event->start);
$date = date('F d, Y', $ts);
$start = date('g:ia', $ts);
$end = date('g:ia', strtotime($event->end));


/*
 * Сгенерировать и возвратить разметку
 */
return "<h2>$event->title</h2>"
    . "\n\t<p class=\"dates\">$date, $start&mdash;$end</p>"
    . "\n\t<p>$event->description</p>";
}

private function _loadEventData($id=NULL) {...}

private function _createEventObj() {...}
```

```
    private function _loadEventById($id) {...}

}

?>
```

### Создание нового файла для отображения подробного описания событий

Для отображения вывода, возвращаемого методом `displayEvent()`, создадим новый файл. Назовем его `view.php` и поместим в папку `public` (`public/view.php`).

Этот файл будет вызываться в строке запроса, содержащей уникальный идентификатор (ID) отображаемого события. Если идентификатор не предоставлен, пользователь будет перенаправлен обратно в основное представление календаря.

В начале файла `view.php` выполняется проверка идентификатора события и загружается файл инициализации, в соответствующие переменные заносятся название страницы и имена CSS-файлов и вызывается файл, отображающий начальную часть страницы, а затем создается новый экземпляр класса `Calendar`.

Далее создается новый дескриптор `div` с идентификатором `content` и вызывается метод `displayEvent`. Затем добавляется ссылка для возврата на основную страницу календаря, закрывается дескриптор `div` и включается файл нижнего колонтитула.

Содержимое файла, реализующего описанные действия, приведено в следующем листинге.

```
<?php

/*
 * Убедиться в том, что ID был передан
 */
if ( isset($_GET['event_id']) )
{
    /*
     * Убедиться в том, что ID является целым числом
     */
    $id = preg_replace('/[^0-9]/', '', $_GET['event_id']);

    /*
     * В случае недействительности ID вернуть пользователя
     * на основную страницу
     */
    if ( empty($id) )
    {
        header("Location: ./");
        exit;
    }
}
else
{
    /*
     * Если ID не был предоставлен, вернуть пользователя
     * на основную страницу
     */
    header("Location: ./");
    exit;
}
```

```
/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Вывести начальную часть страницы
 */
$page_title = "Просмотр событий";
$css_files = array("style.css");
include_once 'assets/common/header.inc.php';

/*
 * Загрузить календарь
 */
 $cal = new Calendar($dbo);

?>

<div id="content">
<?php echo $cal->displayEvent($id) ?>

    <a href="./">&laquo; Вернуться в календарь</a>
</div><!-- end #content -->

<?php

/*
 * Вывести завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Протестируйте файл, вернувшись в основное представление календаря и щелкнув на названии события. Это должно привести к загрузке файла `view.php` и отображению информации о событии в формате, соответствующем календарю (рис. 4.8).

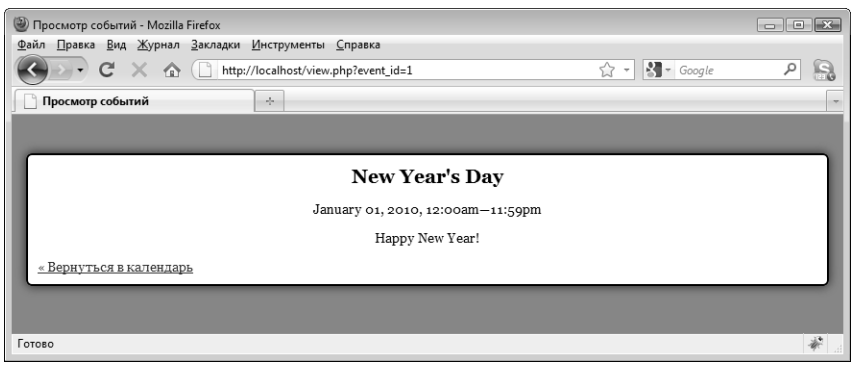



**Рис. 4.8.** Информация о событии, отображаемая после щелчка на его названии

# Резюме

Теперь у вас имеется работающий календарь событий, созданный с использованием PHP и MySQL. В процессе его создания вы научились работать с датами, организовывать данные внутри объектов для упрощения доступа к ним и выводить HTML-разметку календаря с использованием таблиц стилей для придания ему традиционного внешнего вида. В следующей главе мы займемся созданием элементов управления, обеспечивающих возможность создания, изменения и удаления событий.

# Глава 5

# Добавление элементов управления для создания, редактирования и удаления событий

Теперь, когда календарь уже можно просматривать, в него необходимо добавить элементы управления, с помощью которых пользователи, наделенные административными полномочиями, могли бы создавать, редактировать и удалять события.

## Генерация формы для создания и редактирования событий

Чтобы редактировать существующие события или добавлять новые, требуется форма. Для генерации такой формы добавим в класс `Calendar` метод `displayForm()`.

Этот простой метод решает следующие задачи:

- проверяет, передан ли ему целочисленный параметр, представляющий уникальный идентификатор события;
- создает пустые переменные для различных полей, описывающих событие;
- загружает информацию о событии, если был передан соответствующий идентификатор;
- сохраняет в ранее созданных переменных информацию о событии, если оно существует;
- отображает форму.

**Примечание.** Явная проверка и, при необходимости, коррекция идентификатора события, передаваемого в суперглобальной переменной `$_POST`, обеспечивают безопасность его использования, поскольку любое нецелочисленное значение будет преобразовано в `0`.

Создайте метод `displayForm()`, добавив в класс `Calendar` следующий код, выделенный ниже полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;
```

```
private $_m;

private $_y;

private $_daysInMonth;

private $_startDay;

public function __construct($dbo=NULL, $useDate=NULL) {...}

public function buildCalendar() {...}

public function displayEvent($id) {...}

/**
 * Генерирует форму, позволяющую редактировать или
 * создавать события
 *
 * @return string: HTML-разметка формы для
 * редактирования событий
 */
public function displayForm()
{
    /*
     * Проверить, был ли передан идентификатор (ID)
     */
    if ( isset($_POST['event_id']) )
    {
        $id = (int) $_POST['event_id'];
            // Принудительно задать целочисленный тип для
            // обеспечения безопасности данных
    }
    else
    {
        $id = NULL;
    }

    /*
     * Инициализировать переменную, хранящую текст заголовка и
     * надписи на кнопке отправки формы
     */
    $submit = "Создать событие";

    /*
     * Если передан ID, загрузить соответствующее событие
     */
    if ( !empty($id) )
    {
        $event = $this->_loadEventById($id);

        /*
         * Если не возвращен объект, возвратить NULL
         */
        if ( !is_object($event) ) { return NULL; }
```

```
            $submit = "Изменить событие";
        }

        /*
         * Создать разметку
         */
        return <<<FORM_MARKUP

    <form action="assets/inc/process.inc.php" method="post">
        <fieldset>
            <legend>$submit</legend>
            <label for="event_title">Название события</label>
            <input type="text" name="event_title"
                  id="event_title" value="$event->title" />
            <label for="event_start">Время начала</label>
            <input type="text" name="event_start"
                  id="event_start" value="$event->start" />
            <label for="event_end">Время окончания</label>
            <input type="text" name="event_end"
                  id="event_end" value="$event->end" />
            <label for="event_description">Описание события</label>
            <textarea name="event_description"
                  id="event_description">$event->description
            </textarea>
            <input type="hidden" name="event_id" value="$event->id" />
            <input type="hidden" name="token" value="$_SESSION[token]" />
            <input type="hidden" name="action" value="event_edit" />
            <input type="submit" name="event_submit" value="$submit" />
            or <a href="./">cancel</a>
        </fieldset>
    </form>
FORM_MARKUP;
    }

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

}

?>
```

## Добавление маркера в форму

Если посмотреть на приведенный выше код формы, то в нем можно увидеть скрытый элемент ввода `token`, содержащий *маркер сеанса* (token). Это мера безопасности, направленная на предотвращение *атак путем подделки межсайтовых запросов* (cross-site request forgeries, CSRF), суть которых заключается в отправке формы обрабатывающему файлу приложения с постороннего сайта, не имеющего никакого отношения к сайту, с которого была загружена форма. Такую тактику спамеры часто применяют для рассылки множественных фальшивых запросов, представляю-

щих собой потенциальную угрозу безопасности, раздражающих пользователей и явно нежелательных.

Маркер защиты от CSRF создается путем генерации случайного хеш-кода и сохранения его в сеансе для последующей отправки вместе с данными формы. Совпадение значения маркера, хранящегося в суперглобальной переменной `$_POST`, со значением, хранящимся в суперглобальной переменной `$_SESSION`, может считаться надежной гарантией достоверности запроса.

Чтобы включить маркер в приложение, добавьте в файл инициализации код, выделенный ниже полужирным шрифтом.

```
<?php

/*
 * Запуск сеанса
 */

session_start();

/*
 * Сгенерировать маркер защиты от CSRF, если это не было
 * сделано ранее
 */
if ( !isset($_SESSION['token']) )
{
    $_SESSION['token'] = sha1(uniqid(mt_rand(), TRUE));
}

/*
 * Включить необходимую конфигурационную информацию
 */
include_once '../sys/config/db-cred.inc.php'; // Информация о БД

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать PDO-объект
 */
$dsn = "mysql:host=" . DB_HOST . ";dbname=" . DB_NAME;
$dbo = new PDO($dsn, DB_USER, DB_PASS);

/*
 * Определить для классов функцию автозагрузки
 */
function __autoload($class)
{
    $filename = "../sys/class/class." . $class . ".inc.php";
    if ( file_exists($filename) )
    {
```

```
        include_once $filename;
    }
}

?>
```

**Предупреждение.** В качестве дополнительной меры безопасности целесообразно ограничить время жизни маркера. Например, проверка того, что маркер существует не более 20 минут, поможет снизить вероятность хищения данных злоумышленниками, если пользователь оставит компьютер на некоторое время без присмотра. Для получения более подробной информации относительно маркеров и методов борьбы с CSRF посетите блог Криса Шифлетта (Chris Shiflett) и ознакомьтесь с его статьей на эту тему по такому адресу `http://shiflett.org/csrf`

## Создание файла для отображения формы

Теперь, имея метод для отображения формы, создадим файл, вызывающий этот метод. Присвоим файлу имя `admin.php` и поместим его в папку `public` (`/public/admin.php`).

Аналогично файлу `view.php`, этот файл решает следующие задачи:

- загружает файл инициализации;
- устанавливает название страницы и массив CSS-файлов;
- включает в вывод начальную часть страницы;
- создает новый экземпляр класса `Calendar`;
- вызывает метод `displayForm()`;
- включает в вывод завершающую часть страницы.

Введите в новый файл `admin.php` следующий код.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Вывести начальную часть страницы
 */
$page_title = "Добавление/редактирование события";
$css_files = array("style.css");
include_once 'assets/common/header.inc.php';

/*
 * Загрузить календарь
 */
$cal = new Calendar($dbo);

?>

<div id="content">

<?php echo $cal->displayForm(); ?>
```

```
</div><!-- end #content -->

<?php

/*
 * Вывести завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохранив этот код, выполните в браузере переход по адресу `http://localhost/admin.php`, что позволит увидеть результирующую форму (рис. 5.1).

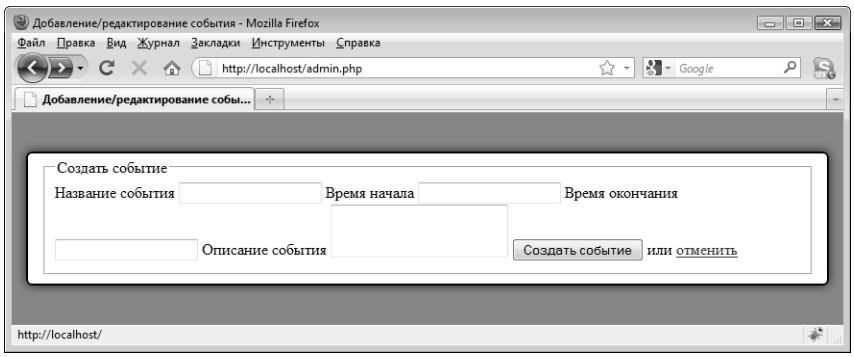



**Рис. 5.1.** Внешний вид формы до применения CSS-стилей

## Добавление новой таблицы стилей для средств администрирования

Очевидно, что внешний вид созданной выше формы нуждается в улучшении. В то же время доступ к ней будут иметь только администраторы (ведь вряд ли вы захотите, чтобы возможность внесения изменений в календарь предоставлялась любому пользователю), поэтому мы выделим соответствующие CSS-правила в отдельную таблицу стилей, которая будет храниться в файле `admin.css` в папке `css` (`/public/assets/css/`).

Следует вновь подчеркнуть, что никаких попыток пояснить использование этих правил здесь не делается, поскольку таблицы CSS не являются предметом рассмотрения данной книги. В сущности, мы используем CSS лишь для того, чтобы придать календарю его привычный вид; кроме того, в данный файл включено также несколько правил, предназначенных для элементов, которые будут созданы позже.

Добавьте в файл `admin.css` следующий код.

```
fieldset {
    border: 0;
}

legend {
    font-size: 24px;
    font-weight: bold;
}
```

```
input[type=text],input[type=password],label {
    display: block;
    width: 70%;
    font-weight: bold;
}

textarea {
    width: 99%;
    height: 200px;
}

input[type=text],input[type=password],textarea {
    border: 1px solid #123;
    -moz-border-radius: 6px;
    -webkit-border-radius: 6px;
    border-radius: 6px;
    -moz-box-shadow: inset 1px 2px 4px #789;
    -webkit-box-shadow: inset 1px 2px 4px #789;
    box-shadow: inset 1px 2px 4px #789;
    padding: 4px;
    margin: 0 0 4px;
    font-size: 16px;
    font-family: georgia, serif;
}

input[type=submit] {
    margin: 4px 0;
    padding: 4px;
    border: 1px solid #123;
    -moz-border-radius: 6px;
    -webkit-border-radius: 6px;
    border-radius: 6px;
    -moz-box-shadow: inset -2px -1px 3px #345,
        inset 1px 1px 3px #BCF,
        1px 2px 6px #789;
    -webkit-box-shadow: inset -2px -1px 3px #345,
        inset 1px 1px 3px #BCF,
        1px 2px 6px #789;
    box-shadow: inset -2px -1px 3px #345,
        inset 1px 1px 3px #BCF,
        1px 2px 6px #789;
    background-color: #789;
    font-family: georgia, serif;
    text-transform: uppercase;
    font-weight: bold;
    font-size: 14px;
    text-shadow: 0px 0px 1px #fff;
}

.admin-options {
    text-align: center;
}

.admin-options form,.admin-options p {
```

```
    display: inline;
}

a.admin {
    display: inline-block;
    margin: 4px 0;
    padding: 4px;
    border: 1px solid #123;
    -moz-border-radius: 6px;
    -webkit-border-radius: 6px;
    border-radius: 6px;
    -moz-box-shadow: inset -2px -1px 3px #345,
        inset 1px 1px 3px #BCF,
        1px 2px 6px #789;
    -webkit-box-shadow: inset -2px -1px 3px #345,
        inset 1px 1px 3px #BCF,
        1px 2px 6px #789;
    box-shadow: inset -2px -1px 3px #345,
        inset 1px 1px 3px #BCF,
        1px 2px 6px #789;
    background-color: #789;
    color: black;
    text-decoration: none;
    font-family: georgia, serif;
    text-transform: uppercase;
    font-weight: bold;
    font-size: 14px;
    text-shadow: 0px 0px 1px #fff;
}
```

Сохраните файл, а затем добавьте запись `admin.css` в массив `$css_files` в файле `admin.php`, внеся для этого изменения, выделенные ниже полужирным шрифтом.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Вывести начальную часть страницы
 */
$page_title = "Добавление/редактирование события";
$css_files = array("style.css", "admin.css");
include_once 'assets/common/header.inc.php';

/*
 * Загрузить календарь
 */
$cal = new Calendar($dbo);

?>

<div id="content">
```

```
<?php echo $cal->displayForm(); ?>

</div><!-- end #content -->

<?php

/*
 * Вывести завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохраните этот код и перезагрузите страницу `http://localhost/admin.php`, что позволит увидеть, как выглядит форма после применения стилей (рис. 5.2).

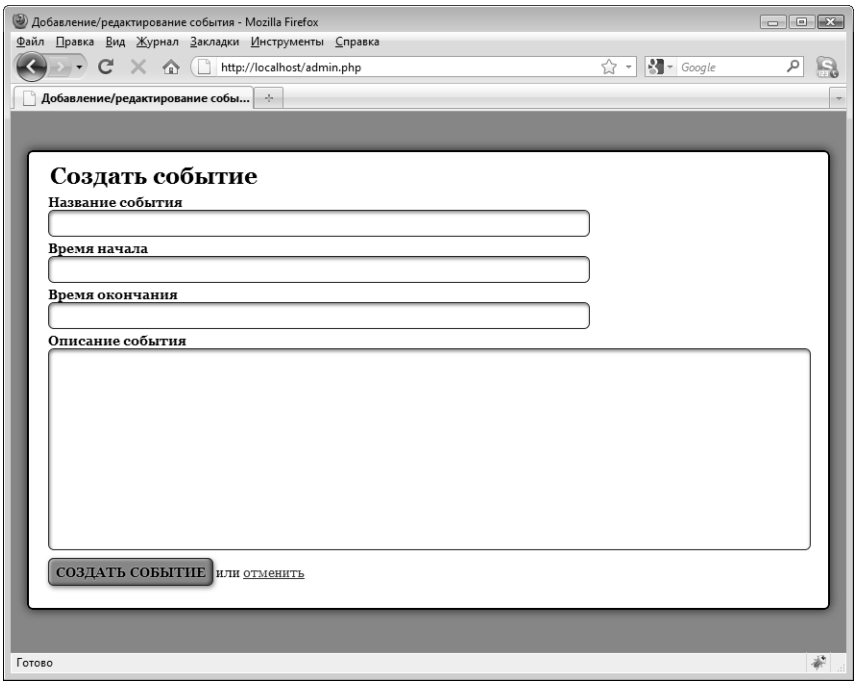



**Рис. 5.2.** Внешний вид формы для добавления или редактирования событий после применения стилей CSS

# Сохранение новых событий в базе данных

Для сохранения событий, вводимых в форму, создадим в классе `Calendar` новый метод `processForm()`, который решает следующие задачи:

- проверяет и при необходимости корректирует данные, переданные с формой с использованием метода POST;
- определяет тип предстоящей операции — редактирование или создание события;

- генерирует инструкцию `INSERT`, если редактируемое событие отсутствует, или `UPDATE`, если был передан уникальный идентификатор события;
- создает предварительно подготовленный запрос и осуществляет привязку параметров;
- выполняет запрос и возвращает `TRUE` или, в случае неудачного завершения, сообщение об ошибке.

Введите код метода `processForm()`, выделенный ниже полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    public function buildCalendar() {...}

    public function displayEvent($id) {...}

    public function displayForm() {...}

    /**
     * Предназначен для проверки формы и сохранения или
     * редактирования события
     *
     * @return mixed: TRUE в случае успешного завершения или
     *                сообщение об ошибке в случае сбоя
     */
    public function processForm()
    {
        /*
         * Выход, если значение "action" задано неправильно
         */
        if ( $_POST['action']!='event_edit' )
        {
            return "Некорректная попытка вызова метода processForm";
        }

        /*
         * Извлечь данные из формы
         */
        $title = htmlentities($_POST['event_title'], ENT_QUOTES);
        $desc = htmlentities($_POST['event_description'], ENT_QUOTES);
```

```
        $start = htmlentities($_POST['event_start'], ENT_QUOTES);
        $end = htmlentities($_POST['event_end'], ENT_QUOTES);

        /*
         * Если ID не был передан, создать новое событие
         */
        if ( empty($_POST['event_id']) )
        {
            $sql = "INSERT INTO `events`
                        (`event_title`, `event_desc`, `event_start`,
                         `event_end`)
                    VALUES
                        (:title, :description, :start, :end)";
        }

        /*
         * Обновить событие, если оно редактировалось
         */
        else
        {
            /*
             * Привести ID события к целочисленному типу в интересах
             * безопасности
             */
            $id = (int) $_POST['event_id'];
            $sql = "UPDATE `events`
                    SET
                        `event_title`=:title,
                        `event_desc`=:description,
                        `event_start`=:start,
                        `event_end`=:end
                    WHERE `event_id`=$id";
        }

        /*
         * После привязки данных выполнить запрос создания или
         * редактирования события
         */
        try
        {
            $stmt = $this->db->prepare($sql);
            $stmt->bindParam(":title", $title, PDO::PARAM_STR);
            $stmt->bindParam(":description", $desc, PDO::PARAM_STR);
            $stmt->bindParam(":start", $start, PDO::PARAM_STR);
            $stmt->bindParam(":end", $end, PDO::PARAM_STR);
            $stmt->execute();
            $stmt->closeCursor();
            return TRUE;
        }
        catch ( Exception $e )
        {
            return $e->getMessage();
        }
    }
```

```
    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

}

?>
```

## Добавление файла, осуществляющего вызов запрошенного метода обработки формы

Форма, предназначенная для добавления и редактирования событий, будет направляться файлу `process.inc.php`, который следует поместить в папку `inc` (`/public/assets/inc/process.inc.php`). Этот файл осуществляет проверку данных, переданных с формой, а также сохраняет или обновляет записи, выполняя для этого следующие действия.

1. Запускает сеанс.
2. Включает файлы, содержащие учетные данные пользователя БД и класс `Calendar`.
3. Определяет константы (перечисленные в файле инициализации).
4. Создает массив, хранящий перечень действий, которые могут выполняться над формой.
5. Проверяет, был ли передан маркер сеанса и, если это так, корректен ли он, а также убеждается в том, что запрашиваемое действие существует в поисковом массиве. В случае положительного результата проверки выполняется п. 6, в противном случае — п. 7.
6. Создает новый экземпляр класса `Calendar`:
   - вызывает метод `processForm()`;
   - возвращает пользователя на основную страницу или выводит сообщение об ошибке в случае неудачного завершения.
7. Возвращает пользователя на основную страницу, не предпринимая никаких действий, если маркер не соответствует заданному.

Создание массива в п. 4 позволяет избежать повторения длинных условных блоков `if...elseif` для последовательного перебора всего списка возможных действий. Использование имен действий как ключей массива, значениями элементов которого являются объекты, методы и страницы, на которые следует перенаправлять пользователя, позволяет создать единый блок проверок, делающий код более компактным и понятным.

Для выполнения описанных выше действий добавьте в файл `process.inc.php` следующий код, выделенный полужирным шрифтом.

```
<?php

/*
 * Запуск сеанса
 */
session_start();
```

```
/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_edit' => array(
                'object' => 'Calendar',
                'method' => 'processForm',
                'header' => 'Location: ../../'
            )
    );

/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( $_POST['token']==$_SESSION['token']
     && isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);
    if ( TRUE === $msg=$obj->$use_array['method']() )
    {
        header($use_array['header']);
        exit;
    }
    else
    {
        // В случае ошибки вывести сообщение о ней и прекратить выполнение
        die ( $msg );
    }
}
else
{
    // В случае некорректности маркера/действия перенаправить
    // пользователя на основную страницу
    header("Location: ../../");
    exit;
}

function __autoload($class_name)
{
```

```
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}

?>
```

Сохраните этот файл, перейдите на страницу `http://localhost/admin.php` и создайте новое событие, используя следующую информацию:

- **название события** — `Dinner Party`;
- **время начала** — `2010-01-22 17:00:00`;
- **время окончания** — `2010-01-22 19:00:00`;
- **описание события** — `Five-course meal with wine pairings at John's house.`

После щелчка на кнопке Создать событие календарь обновится и в нем отобразится новое событие (рис. 5.3).

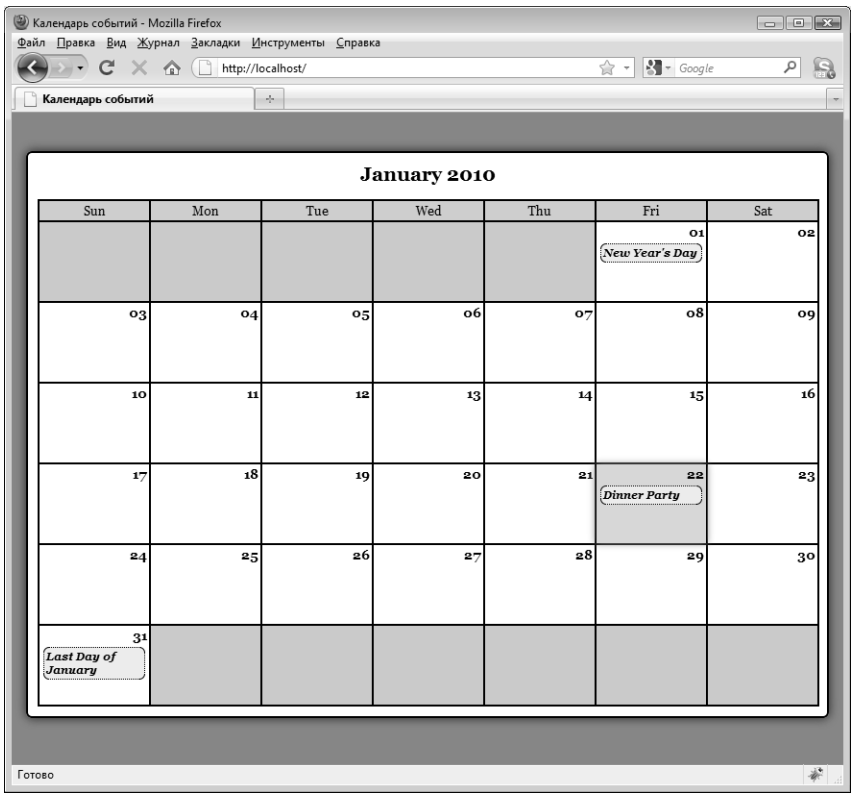



**Рис. 5.3.** Отображение нового события (подсвечено в результате наведения на него указателя)

## Добавление кнопки создания новых событий в основное представление

Чтобы упростить авторизованным пользователям создание новых событий, добавим в календарь кнопку, щелчок на которой вызывает переход на страницу `admin.php`. Для этого создайте в классе `Calendar` новый закрытый метод `_adminGeneralOptions()`.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    public function buildCalendar() {...}

    public function displayEvent($id) {...}

    public function displayForm() {...}

    public function processForm() {...}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

    /**
     * Генерирует разметку для отображения административных ссылок
     *
     * @return string: разметка для отображения административных ссылок
     */
    private function _adminGeneralOptions()
    {
        /*
         * Отобразить административные элементы управления
         */
        return <<<ADMIN_OPTIONS
    <a href="admin.php" class="admin">+ Добавить новое событие</a>
ADMIN_OPTIONS;
    }

}

?>
```

**Примечание.** Необходимые проверки, гарантирующие отображение данной кнопки лишь для авторизованных пользователей, будут добавлены в главе 6.

После этого внесите в метод `buildCalendar()` изменения, обеспечивающие вызов метода `_adminGeneralOptions()`, добавив следующий код, выделенный полужирным шрифтом.

```
public function buildCalendar()
{
    // Для экономии места основная часть кода этого метода опущена

    /*
     * Закрыть окончательный неупорядоченный список
     */
    $html .= "\n\t</ul>\n\n";

    /*
     * Если выполнен вход, отобразить опции администрирования
     */
    $admin = $this->_adminGeneralOptions();

    /*
     * Возвратить разметку для вывода
     */
    return $html . $admin;
}
```

Наконец, чтобы обеспечить корректное отображение ссылки, добавьте в файл `index.php` таблицу стилей для элементов административного управления (`admin.css`), введя код, выделенный ниже полужирным шрифтом:

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь для января
 */
 $cal = new Calendar($dbo, "2010-01-01 12:00:00");

/*
 * Задать название страницы и файлы CSS
 */
$page_title = "Календарь событий";
$css_files = array('style.css', 'admin.css');

/*
 * Включить начальную часть страницы
 */
include_once 'assets/common/header.inc.php';

?>
```

```
<div id="content">

<?php

/*
 * Отобразить календарь в виде HTML
 */
echo $cal->buildCalendar();

?>

</div><!-- end #content -->
<?php

/*
 * Включить завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

После сохранения файла и перезагрузки страницы `http://localhost/` на ней отобразится кнопка (рис. 5.4).

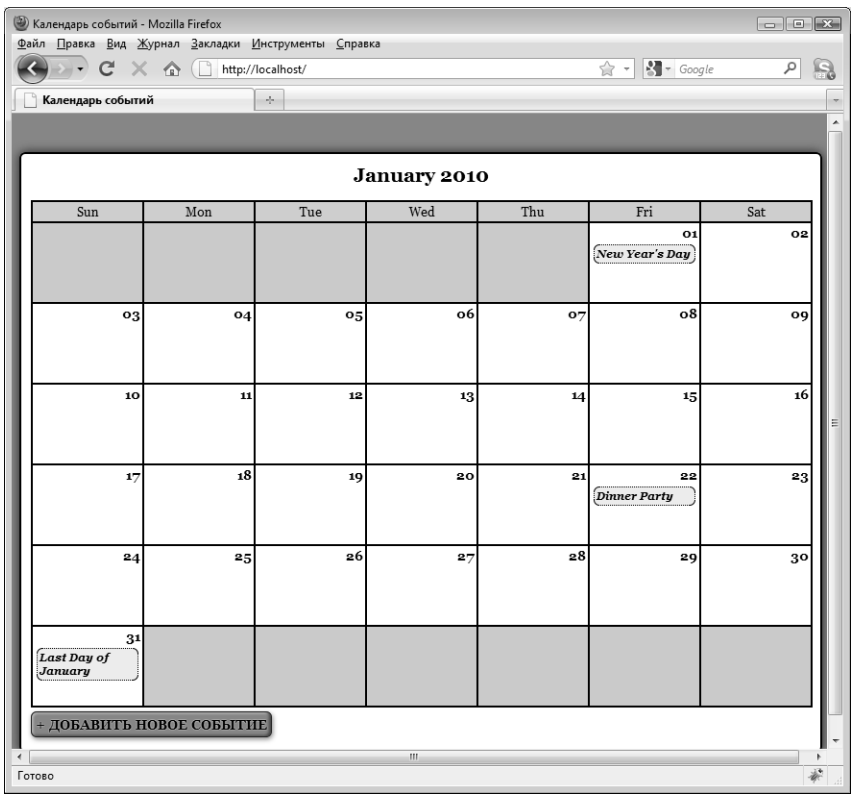



**Рис. 5.4.** Отображение кнопки Добавить новое событие в левом нижнем углу календаря

# Добавление элементов редактирования в подробное представление событий

Далее необходимо предоставить возможность редактирования событий авторизованным пользователям. Мы сделаем это, добавив в файл `view.php` кнопку для отображения подробной информации о событии.

В то же время, в отличие от простых ссылок, используемых для создания новых опций, для кнопки редактирования потребуется фактическая отправка формы. Для повышения управляемости этого кода мы создадим в классе `Calendar` новый закрытый метод `_adminEntryOptions()`, генерирующий разметку для формы.

Пока что эта форма будет просто возвращать разметку для отображения кнопки редактирования. По мере продолжения работы с упражнениями форма будет дополняться новыми элементами.

Соответствующие изменения, которые необходимо внести в класс `Calendar`, выделены в приведенном ниже листинге полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    public function buildCalendar() {...}

    public function displayEvent($id) {...}

    public function displayForm() {...}

    public function processForm() {...}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

    private function _adminGeneralOptions() {...}

    /**
     * Генерирует опции редактирования и удаления для
     * события с заданным идентификатором (ID)
```

```
     *
     * @param int $id: идентификатор события, для которого
     *                 генерируются опции
     * @return string: разметка для опций редактирования/удаления
     */
    private function _adminEntryOptions($id)
    {
        return <<<ADMIN_OPTIONS

    <div class="admin-options">
    <form action="admin.php" method="post">
        <p>
            <input type="submit" name="edit_event"
                  value="Редактировать событие" />
            <input type="hidden" name="event_id"
                  value="$id" />
        </p>
    </form>
    </div><!-- end .admin-options -->
ADMIN_OPTIONS;
    }
}

?>
```

## Модификация метода `displayEvent` для отображения элементов административного управления

Прежде чем будет отображена кнопка редактирования, необходимо вызвать метод `_adminEntryOptions()` из метода `displayEvent()`. Это сводится к сохранению возвращаемого значения метода `_adminEntryOptions()` в переменной `$admin` и выводу этой переменной вместе с остальной частью разметки.

Внесите в метод `displayEvent()` класса `Calendar` изменения, выделенные ниже полужирным шрифтом.

```
    /**
     * Отображает информацию о заданном событии
     *
     * @param int $id: идентификатор (ID) события
     * @return string: элементарная разметка для отображения
     *                 информации о событии
     */
    public function displayEvent($id)
    {
    /*
     * Убедиться в том, что ID был передан
     */
    if ( empty($id) ) { return NULL; }

    /*
     * Проверить, что ID является целым числом
     */
    $id = preg_replace('/[^0-9]/', '', $id);
```

```
    /*
     * Загрузить данные о событии из БД
     */
    $event = $this->_loadEventById($id);

    /*
     * Сгенерировать строки для даты, начального и конечного времени
     */
    $ts = strtotime($event->start);
    $date = date('F d, Y', $ts);
    $start = date('g:ia', $ts);
    $end = date('g:ia', strtotime($event->end));

    /*
     * Загрузить административные опции, если пользователь
     ? выполнил вход
     */
    $admin = $this->_adminEntryOptions($id);

    /*
     * Сгенерировать и возвратить разметку
     */
    return "<h2>$event->title</h2>"
        . "\n\t<p class=\"dates\">$date, $start&mdash;$end</p>"
        . "\n\t<p>$event->description</p>$admin";
    }
```

**Примечание.** Обратите внимание на добавление переменной `$admin` в конце возвращаемой строки.

**Примечание.** Как и в случае кнопки Создать событие, далее будут дополнительно введены проверки, гарантирующие, что только уполномоченные пользователи смогут видеть элементы управления редактированием.

## Добавление административной таблицы стилей в подробное представление событий

Последнее, что нам осталось сделать для того, чтобы кнопкой редактирования можно было пользоваться, — это включить таблицу стилей `admin.css` в переменную `$css_files` в файле `view.php`.

```
<?php

/*
 * Убедиться в том, что ID был передан
 */
if ( isset($_GET['event_id']) )
{
    /*
     * Убедиться в том, что ID является целым числом
     */
    $id = preg_replace('/[^0-9]/', '', $_GET['event_id']);

    /*
```

```
     * В случае недействительности ID вернуть пользователя
     * на основную страницу
     */
    if ( empty($id) )
    {
        header("Location: ./");
        exit;
    }
}
else
{
    /*
     * Если ID не был предоставлен, вернуть пользователя
     * на основную страницу
     */
    header("Location: ./");
    exit;
}

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Вывести начальную часть страницы
 */
$page_title = "Просмотр события";
$css_files = array("style.css", "admin.css");
include_once 'assets/common/header.inc.php';

/*
 * Загрузить календарь
 */
 $cal = new Calendar($dbo);

?>

<div id="content">
<?php echo $cal->displayEvent($id) ?>

    <a href="./">&laquo; Вернуться в календарь</a>
</div><!-- end #content -->

<?php

/*
 * Вывести завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохраните файл, а затем щелкните на событии, чтобы увидеть кнопку редактирования (рис. 5.5).

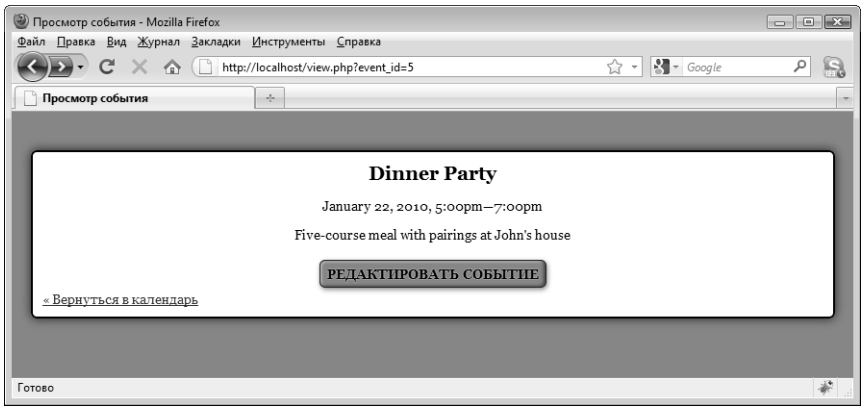



**Рис. 5.5.** Отображение кнопки редактирования при просмотре подробного описания события

Щелчок на кнопке редактирования открывает страницу `admin.php`, в которой все данные загружены в форму (рис. 5.6).

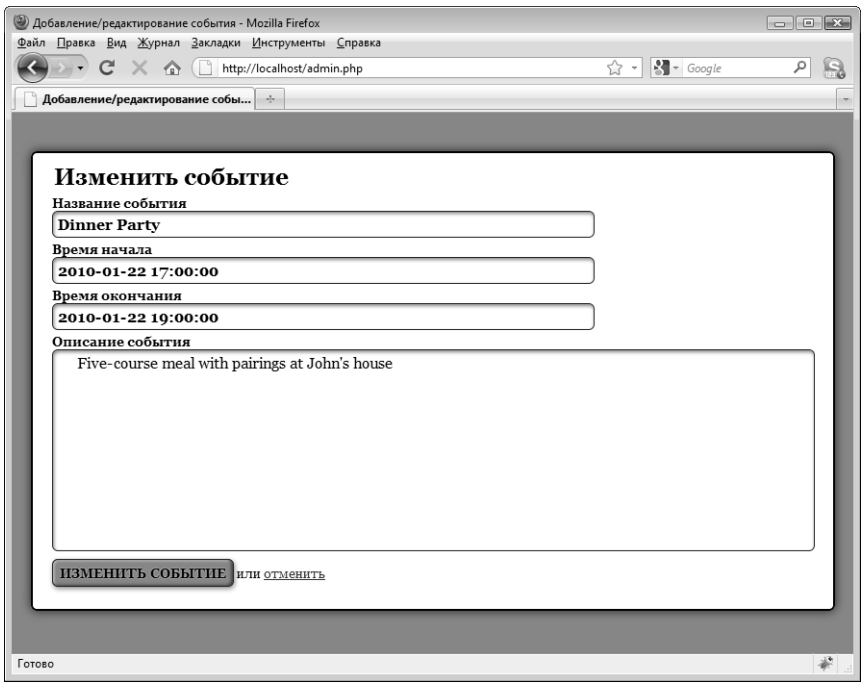



**Рис. 5.6.** Административная форма, отображаемая при редактировании события

# Удаление событий

Последний этап создания календаря заключается в предоставлении возможности удаления событий авторизованным пользователям. Удаление событий отличается от их редактирования или создания тем, что в данном случае целесообразно потребовать от пользователя подтверждения своих намерений. В противном случае нечаянный щелчок может лишить пользователя части информации, доставив ему неудобства.

Таким образом, процедура удаления события должна осуществляться в два этапа.

1. Пользователь выполняет щелчок на кнопке и попадает на страницу, где должен подтвердить свои действия.
2. Чтобы удалить событие, необходимо выполнить щелчок на кнопке подтверждения.

## Генерация кнопки удаления события

Начнем с добавления кнопки удаления событий в представление подробного отображения информации. Для этого внесите в метод `_adminEntryOptions()` класса `Calendar` изменения, выделенные ниже полужирным шрифтом.

```
    /**
     * Генерирует опции редактирования и удаления для
     * события с заданным идентификатором (ID)
     *
     * @param int $id: идентификатор события, для которого
     *                 генерируются опции
     * @return string: разметка для опций редактирования/удаления
     */
    private function _adminEntryOptions($id)
    {
        return <<<ADMIN_OPTIONS

    <div class="admin-options">
    <form action="admin.php" method="post">
        <p>
            <input type="submit" name="edit_event"
                  value="Редактировать событие" />
            <input type="hidden" name="event_id"
                  value="$id" />
        </p>
    </form>
    <form action="confirmdelete.php" method="post">
        <p>
            <input type="submit" name="delete_event"
                  value="Удалить событие" />
            <input type="hidden" name="event_id"
                  value="$id" />
        </p>
    </form>
    </div><!-- end .admin-options -->
ADMIN_OPTIONS;
    }
```

Этот код добавляет кнопку, перенаправляющую пользователя на страницу подтверждения `confirmdelete.php`, которую нам еще предстоит создать. После сохранения указанных изменений вы увидите в представлении подробного отображения событий как кнопку редактирования, так и кнопку удаления событий (рис. 5.7).

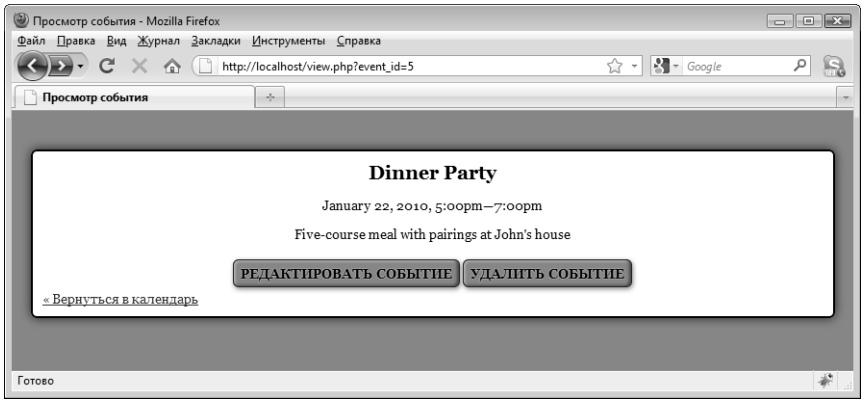



**Рис. 5.7.** Отображение кнопки удаления событий при просмотре их подробных описаний

## Создание метода, запрашивающего подтверждение удаления события

При щелчке на кнопке удаления события пользователь попадает на страницу подтверждения, отображающую форму, на которой он может подтвердить свои намерения. Для генерации этой формы мы создадим в классе `Calendar` новый общедоступный метод `confirmDelete()`.

В процессе получения подтверждения того, что событие действительно следует удалить, метод выполняет следующие действия.

1. Проверяет, была ли форма отправлена и был ли передан действительный маркер сеанса. Если эти условия выполняются, осуществляется переход к п. 2, иначе — к п. 3.
2. Проверяет, был ли щелчок выполнен именно на кнопке подтверждения:
   - если это так, то событие удаляется;
   - в противном случае пользователь перенаправляется обратно в основное представление календаря.
3. Загружает информацию о событии и отображает форму на странице подтверждения.

Описанные действия обеспечиваются кодом нового метода, выделенным в приведенном ниже листинге полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;
```

```
private $_m;

private $_y;

private $_daysInMonth;

private $_startDay;

public function __construct($dbo=NULL, $useDate=NULL) {...}

public function buildCalendar() {...}

public function displayEvent($id) {...}

public function displayForm() {...}

public function processForm() {...}

/**
 * Получает подтверждение необходимости удаления события
 * и, если это действительно требуется, выполняет операцию.
 *
 * После выполнения пользователем щелчка на кнопке удаления
 * генерирует диалоговое окно подтверждения. При получении
 * подтверждения удаляет событие из базы данных и отправляет
 * пользователя обратно в основное представление календаря. Если
 * пользователь передумал удалять событие, он возвращается назад
 * в основное представление календаря, и удаления события
 * не происходит.
 *
 * @param int $id: идентификатор (ID) события
 * @return mixed: форма в случае получения подтверждения и
 * ничего или сообщение об ошибке в случае удаления события
 */
public function confirmDelete($id)
{
    /*
     * Убедиться в том, что идентификатор (ID) был передан
     */
    if ( empty($id) ) { return NULL; }

    /*
     * Убедиться в том, что идентификатор является целым числом
     */
    $id = preg_replace('/[^0-9]/', '', $id);

    /*
     * Если была отправлена форма подтверждения, снабженная
     * действительным маркером, проверить данные, переданные
     * вместе с формой
     */
    if ( isset($_POST['confirm_delete'])
         && $_POST['token']==$_SESSION['token'] )
    {
```

```
        /*
         * Если удаление подтверждено пользователем, удалить
         * событие из базы данных
         */
        if ( $_POST['confirm_delete']=="Да, удалить" )
        {
            $sql = "DELETE FROM `events`
                    WHERE `event_id`=:id
                    LIMIT 1";
            try
            {
                $stmt = $this->db->prepare($sql);
                $stmt->bindParam(
                      ":id",
                      $id,
                      PDO::PARAM_INT
                    );
                $stmt->execute();
                $stmt->closeCursor();
                header("Location: ./");
                return;
            }
            catch ( Exception $e )
            {
                return $e->getMessage();
            }
        }

        /*
         * Если удаление не подтверждено пользователем, вернуть
         * его на основную страницу
         */
        else
        {
            header("Location: ./");
            return;
        }
    }

    /*
     * Если форма для подтверждения не была передана,
     * отобразить ее
     */
    $event = $this->_loadEventById($id);

    /*
     * Если не был возвращен объект, вернуться в основное
     * представление
     */
    if ( !is_object($event) ) { header("Location: ./"); }

    return <<<CONFIRM_DELETE

<form action="confirmdelete.php" method="post">
```

```
        <h2>
            Вы действительно хотите удалить событие "$event->title"?
        </h2>
        <p><strong>Удаленное событие <strong>невозможно восстановить</strong></
p>
        <p>
            <input type="submit" name="confirm_delete"
                  value="Да, удалить" />
            <input type="submit" name="confirm_delete"
                  value="Нет! Это была шутка!" />
            <input type="hidden" name="event_id"
                  value="$event->id" />
            <input type="hidden" name="token"
                  value="$_SESSION[token]" />
        </p>
    </form>
CONFIRM_DELETE;
    }

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

    private function _adminGeneralOptions() {...}

    private function _adminEntryOptions($id) {...}

}

?>
```

## Создание файла для отображения подтверждающей формы

Для вызова метода `confirmDelete()` создадим файл `confirmdelete.php` и поместим его в папку `public` (`/public/confirmdelete.php`). По своему назначению этот файл аналогичен файлу `index.php` и решает следующие задачи:

- проверяет, был ли передан уникальный идентификатор (ID) события, и сохраняет его в переменной `$id`; если идентификатор не был передан, перенаправляет пользователя обратно в основное представление календаря;
- загружает файл инициализации;
- создает новый экземпляр класса `Calendar`;
- загружает возвращенное методом `confirmDelete()` значение в переменную `$markup`;
- определяет значения переменных `$page_title` и `$css_files` и включает начальную часть страницы;
- выводит данные, сохраненные в переменной `$markup`;
- выводит завершающую часть страницы.

**Примечание.** Причина, по которой включению начальной части страницы предшествует загрузка возвращенного методом `confirmDelete()` значения в переменную, состоит в том, что этот метод иногда использует функцию `header()` для перенаправления пользователя в другие части приложения. Если файл начальной части страницы окажется включенным до вызова метода `confirmDelete()`, то в некоторых случаях при выполнении сценария может наступать сбой, поскольку в браузер не смогут быть выведены никакие данные, прежде чем не будет вызвана функция `header()`, иначе возможен фатальный сбой. Более подробную информацию относительно функции `header()` можно найти по адресу `http://php.net/header`.

А теперь добавьте в файл `confirmdelete.php` следующий код.

```
<?php

/*
 * Убедиться в том, что ID был передан
 */
if ( isset($_POST['event_id']) )
{
    /*
     * Извлечь ID события из строки URL
     */
    $id = (int) $_POST['event_id'];
}
else
{
    /*
     * Если ID не был предоставлен, вернуть пользователя
     * на основную страницу
     */
    header("Location: ./");
    exit;
}

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь
 */
$cal = new Calendar($dbo);
$markup = $cal->confirmDelete($id);

/*
 * Вывести начальную часть страницы
 */
$page_title = "Просмотр события";
$css_files = array("style.css", "admin.css");
include_once 'assets/common/header.inc.php';

?>

<div id="content">
<?php echo $markup; ?>
```

```
</div><!-- end #content -->

<?php

/*
 * Output the footer
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохраните файл и протестируйте приложение, попытавшись удалить запись “Dinner Party”. После отображения страницы с подробным описанием события календарь перенаправит вас на страницу подтверждения (рис. 5.8).

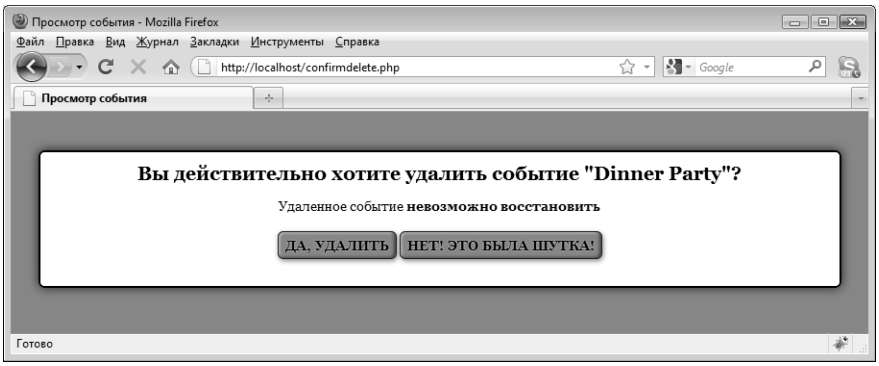



**Рис. 5.8.** Подтверждающая форма, предлагаемая пользователю после щелчка на кнопке удаления события

После щелчка на кнопке Да, удалить событие удаляется из календаря (рис. 5.9).

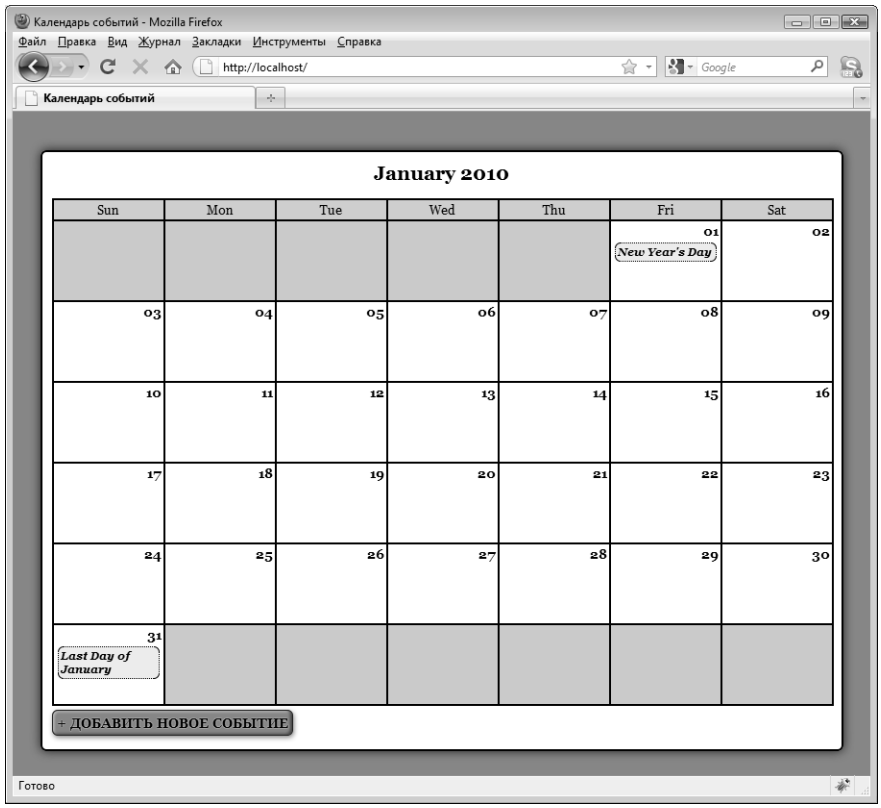



**Рис. 5.9.** После получения подтверждения событие удаляется из календаря

# Резюме

К этому моменту в вашем распоряжении имеется функционирующий календарь событий. Вы узнали о том, как сформировать форму, предназначенную для создания, редактирования, сохранения и удаления событий, и получить от пользователя подтверждение его действий. Однако в настоящее время все имеющиеся элементы административного управления доступны любому посетителю сайта.

В следующей главе мы создадим класс, предоставляющий доступ к элементам административного управления вашего сайта лишь авторизованным пользователям.

# Глава 6

# Парольная защита критических данных и операций над ними

Теперь, когда приложение позволяет добавлять, изменять и удалять события, стоит предусмотреть защиту от несанкционированных действий пользователей, потребовав от них ввода своих учетных данных, прежде чем предоставить им возможность вносить в календарь какие-либо изменения. Для этого потребуется создать новую таблицу базы данных и новый класс, а также внести кое-какие изменения в существующие файлы.

## Создание административной таблицы в базе данных

Для хранения информации о пользователях, имеющих полномочия на изменение информации о событиях, создадим новую таблицу базы данных. В этой таблице, которую мы назовем `users`, для каждого пользователя будут храниться четыре элемента данных: уникальный идентификатор, имя пользователя, хеш-код пароля и адрес электронной почты.

Чтобы создать указанную таблицу, перейдите на страницу `http//localhost/phpmyadmin`, выберите вкладку `SQL` и выполните показанную ниже команду.

```
CREATE TABLE IF NOT EXISTS `php-jquery_example`.`users` (
  `user_id` INT(11) NOT NULL AUTO_INCREMENT,
  `user_name` VARCHAR(80) DEFAULT NULL,
  `user_pass` VARCHAR(47) DEFAULT NULL,
  `user_email` VARCHAR(80) DEFAULT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE (`user_name`)
) ENGINE=MyISAM CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

После того как этот код выполнится, выберите базу данных `php-jquery_example` в левом столбце и щелкните на таблице `users` для ее просмотра (рис. 6.1).

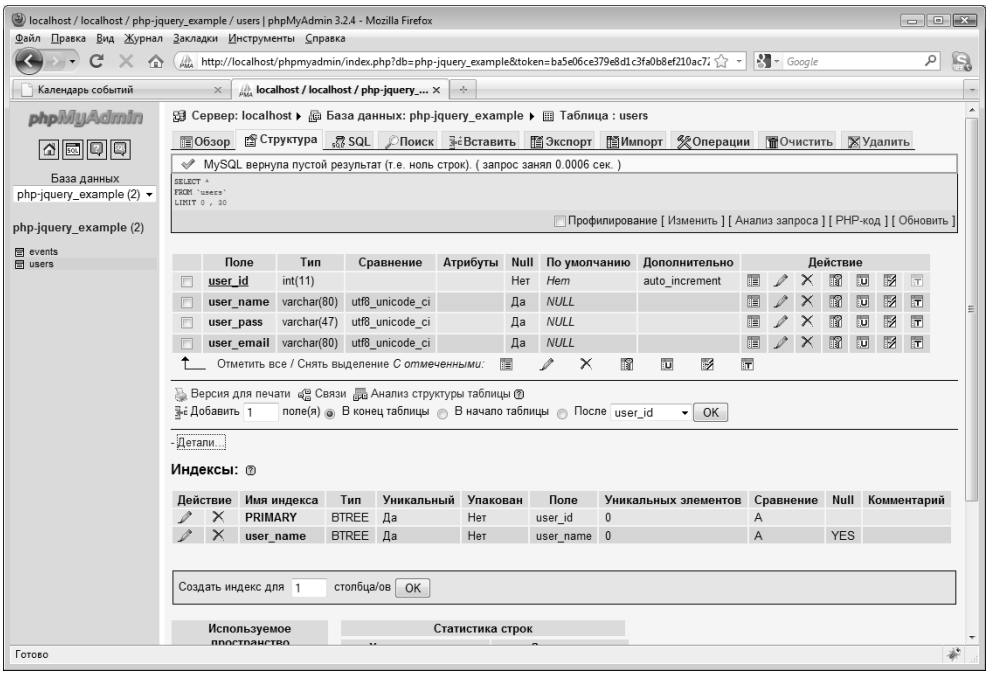



**Рис. 6.1.** Отображение таблицы `users` в окне phpMyAdmin

# Создание файла для отображения регистрационной формы

Чтобы выполнить процедуру входа, пользователь должен получить доступ к регистрационной форме. Для отображения этой формы создадим файл `login.php` и поместим его в папку `public` (`/public/login.php`). Данный файл аналогичен файлу `admin.php`, за исключением того, что он всего лишь выводит форму, поскольку ни один из связанных с ним элементов данных не является переменной.

Форма будет принимать имя пользователя и пароль, а передавать — маркер сеанса и идентификатор вида действия `user_login`. Для создания указанной формы введите в файл `login.php` следующий код.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Вывести начальную часть страницы
 */
$page_title = "Пожалуйста, зарегистрируйтесь";
$css_files = array("style.css", "admin.css");
include_once 'assets/common/header.inc.php';
```

```
?>

<div id="content">

    <form action="assets/inc/process.inc.php" method="post">
        <fieldset>
            <legend>Пожалуйста, зарегистрируйтесь</legend>
            <label for="uname">Имя пользователя</label>
            <input type="text" name="uname"
                   id="uname" value="" />
            <label for="pword">Пароль</label>
            <input type="password" name="pword"
                   id="pword" value="" />
            <input type="hidden" name="token"
                value="<?php echo $_SESSION['token']; ?>" />
            <input type="hidden" name="action"
                value="user_login" />
            <input type="submit" name="login_submit"
                value="Вход" />
            или <a href="./">отменить</a>
        </fieldset>
    </form>

</div><!-- end #content -->

<?php

/*
 * Вывести завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохраните файл и перейдите в браузере на страницу `http://localhost/login.php`, на которой должна отобразиться регистрационная форма для выполнения пользователем процедуры входа (рис. 6.2).

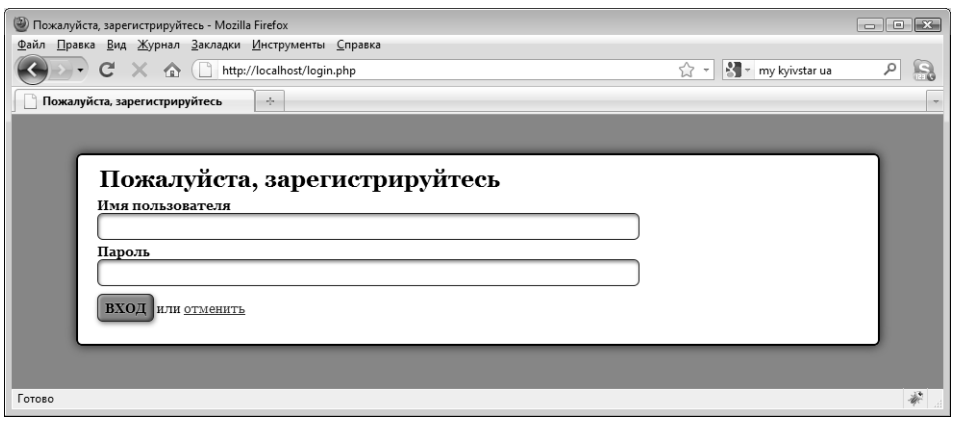



**Рис. 6.2.** Форма для регистрации пользователей

# Создание класса Admin

Подготовив таблицу, можно приступить к построению класса, который будет с ней взаимодействовать. Создайте в папке `class` новый класс `admin.inc.php` (`/sys/class/class.admin.inc.php`). Этот класс будет содержать методы, ответственные за выполнение пользователем процедур входа и выхода.

## Определение класса

Прежде всего, чтобы иметь возможность доступа к базе данных, определите класс, наследующий класс `DB_Connect`. У этого класса будет одно закрытое свойство, `$_saltLength`, о котором мы поговорим немного позднее.

Чтобы гарантировать существование объекта базы данных, конструктор будет вызывать конструктор родительского класса, а затем проверять, передано ли ему в качестве второго аргумента целое число. Если данное условие выполняется, то указанное целочисленное значение присваивается свойству `$_saltLength`.

Далее определите класс, свойство и конструктор, введя в файл `admin.inc.php` следующий код.

```
<?php

/**
 * Управляет выполнением административных задач
 *
 * Версия PHP 5
 *
 * ЛИЦЕНЗИЯ: на этот файл распространяется лицензия MIT,
 * доступная по адресу:
 * http://www.opensource.org/licenses/mit-license.html
 *
 * @author     Jason Lengstorf <jason.lengstorf@ennuidesign.com>
 * @copyright  2009 Ennui Design
 * @license    http://www.opensource.org/licenses/mit-license.html
 */
class Admin extends DB_Connect
{

    /**
     * Определяет длину затравки, используемой при хешировании паролей
     *
     * @var int: длина используемой затравки
     */
    private $_saltLength = 7;

     /**
     * Сохраняет или создает объект БД и устанавливает длину затравки
     *
     * @param object $db: объект базы данных
     * @param int $saltLength: длина затравки для хеширования пароля
     */
    public function __construct($db=NULL, $saltLength=NULL)
    {
        parent::__construct($db);
```

```
        /*
         * Если передан целочисленный параметр, задать длину затравки
         */
        if ( is_int($saltLength) )
        {
            $this->_saltLength = $saltLength;
        }
    }

}

?>
```

## Создание метода для проверки учетных данных пользователя

Чтобы удостовериться в том, что пользователь имеет полномочия для внесения изменений в таблицу `events`, данные, поступающие из файла `login.php`, должны проходить проверку. Для реализации этого можно применить следующую процедуру.

1. Убедиться в том, что с формой был передан подходящий идентификатор вида действия.
2. Осуществить проверку и коррекцию пользовательского ввода с помощью функции `htmlentities()`.
3. Извлечь из базы данных информацию, соответствующую введенному имени пользователя.
4. Сохранить информацию о пользователе в переменной `$user` и убедиться в том, что эта переменная не пуста.
5. Сгенерировать хешированные с использованием *затравки* (примеси) значения пароля, предоставленного пользователем, и пароля, хранящегося в базе данных.
6. Убедиться в совпадении хеш-кодов обоих паролей.
7. Сохранить пользовательские данные в текущем сеансе, воспользовавшись для этого массивом, и вернуть значение `TRUE`.

---

**Примечание.** Хеш-коды с затравкой рассматриваются в следующем разделе.

---

Сначала определим соответствующий метод в классе `Admin` и обеспечим выполнение действий, описанных в пп. 1 и 2, с помощью кода, выделенного ниже полужирным шрифтом.

```
<?php

class Admin extends DB_Connect
{

    private $_saltLength = 7;

    public function __construct($db=NULL, $saltLength=NULL) {...}

    /**
```

```
     * Проверяет действительность учетных данных пользователя
     *
     * @return mixed: TRUE в случае успешного завершения, иначе --
     * сообщение об ошибке
     */
    public function processLoginForm()
    {

        /*
         * Аварийное завершение, если был отправлен недействительный
         * атрибут ACTION
         */
        if ( $_POST['action']!='user_login' )
        {
            return "В processLoginForm передано
              недействительное значение атрибута ACTION";
        }

        /*
         * Маскировать пользовательский ввод в целях безопасности
         */
        $uname = htmlentities($_POST['uname'], ENT_QUOTES);
        $pword = htmlentities($_POST['pword'], ENT_QUOTES);

        // закончить обработку...
    }

}

?>
```

Затем выполните пп. 3 и 4, добавив выделенный ниже код.

```
    public function processLoginForm()
    {

        /*
         * Аварийное завершение, если был отправлен недействительный
         * атрибут ACTION
         */
        if ( $_POST['action']!='user_login' )
        {
            return "В processLoginForm передано
              недействительное значение атрибута ACTION";
        }

        /*
         * Маскировать пользовательский ввод в целях безопасности
         */
        $uname = htmlentities($_POST['uname'], ENT_QUOTES);
        $pword = htmlentities($_POST['pword'], ENT_QUOTES);

        /*
         * Извлечь из базы данных совпадающую информацию,
         * если она существует
```

```
     */
    $sql = "SELECT
                `user_id`, `user_name`, `user_email`, `user_pass`
            FROM `users`
            WHERE
                `user_name` = :uname
            LIMIT 1";
    try
    {
        $stmt = $this->db->prepare($sql);
        $stmt->bindParam(':uname', $uname, PDO::PARAM_STR);
        $stmt->execute();
        $user = array_shift($stmt->fetchAll());
        $stmt->closeCursor();
    }
    catch ( Exception $e )
    {
        die ( $e->getMessage() );
    }

    /*
     * Аварийное завершение, если имя пользователя не
     * согласуется ни с одной записью в БД
     */
    if ( !isset($user) )
    {
        return "Неверное имя пользователя или пароль.";
    }

    // закончить обработку...
}
```

Теперь данные о пользователе сохранены в переменной `$user` (или же метод завершился аварийно из-за того, что для предоставленного имени пользователя не нашлось соответствия в таблице `users`).

Задачи метода исчерпываются выполнением пп. 5–7; для этого дополните код метода выделенным ниже кодом.

```
public function processLoginForm()
{

    /*
     * Аварийное завершение, если был отправлен недействительный
     * атрибут ACTION
     */
    if ( $_POST['action']!='user_login' )
    {
        return "В processLoginForm передано
          недействительное значение атрибута ACTION";
    }

    /*
     * Маскировать пользовательский ввод в целях безопасности
     */
    $uname = htmlentities($_POST['uname'], ENT_QUOTES);
```

```
$pword = htmlentities($_POST['pword'], ENT_QUOTES);

/*
 * Извлечь из базы данных совпадающую информацию,
 * если она существует
 */
$sql = "SELECT
            `user_id`, `user_name`, `user_email`, `user_pass`
        FROM `users`
        WHERE
            `user_name` = :uname
        LIMIT 1";
try
{
    $stmt = $this->db->prepare($sql);
    $stmt->bindParam(':uname', $uname, PDO::PARAM_STR);
    $stmt->execute();
    $user = array_shift($stmt->fetchAll());
    $stmt->closeCursor();
}
catch ( Exception $e )
{
    die ( $e->getMessage() );
}

/*
 * Аварийное завершение, если имя пользователя не
 * согласуется ни с одной записью в БД
 */
if ( !isset($user) )
{
    return "Пользователя с таким именем не существует в БД.";
}

/*
 * Получить хеш-код пароля, предоставленного пользователем
 */
$hash = $this->_getSaltedHash($pword, $user['user_pass']);

/*
 * Проверить, совпадает ли хешированный пароль с
 * сохраненным в БД хеш-кодом
 */
if ( $user['user_pass']==$hash )
{
    /*
     * Сохранить пользовательскую информацию в сеансе
     * в виде массива
     */
    $_SESSION['user'] = array(
            'id' => $user['user_id'],
            'name' => $user['user_name'],
            'email' => $user['user_email']
        );
```

```
            return TRUE;
        }

        /*
         * Аварийное завершение в случае несовпадения паролей
         */
        else
        {
            return "Неверное имя пользователя или пароль.";
        }
    }
```

Теперь данный метод способен осуществлять проверку данных, отправленных с регистрационной формой. Однако для того чтобы он мог работать, требуется создать еще метод `_getSaltedHash()`.

## Метод для генерации хеш-кодов с затравкой

Для выполнения сверки с хеш-кодом пароля, сохраненным в базе данных, необходима функция, которая генерировала бы с помощью затравки хешированное значение пароля, предоставленного пользователем (*хеш-код* — это шифрованная строка, сгенерированная любым безопасным алгоритмом, например MD5 или SHA1).

**Примечание.** Более подробную информацию о хешировании паролей и безопасных криптографических алгоритмах можно найти по следующему адресу:
`http://ru.wikipedia.org/wiki/Криптографическая_хеш-функция`

### Усиление безопасности посредством использования затравки при хешировании паролей

Хотя в PHP и предусмотрены функции, осуществляющие *хеширование* (т.е. шифрование) строк, но для пущей уверенности в том, что данные никто не сможет похитить, следует всегда использовать дополнительные меры безопасности. Одним из простейших и вместе с тем наиболее эффективных способов усиления безопасности является применение *затравки* (salt), которая представляет собой произвольную строку, используемую для получения хеш-кода пароля.

#### Радужные таблицы и обычные алгоритмы шифрования

Обычные алгоритмы шифрования, такие как SHA1 или MD5, взламываются с помощью так называемых *радужных таблиц* (rainbow tables)[1], которые представляют собой таблицы обратного поиска для хеш-кодов паролей. По сути, используя этот метод, хакер может осуществить поиск хеш-кода, полученного с помощью заданного алгоритма шифрования, в большой таблице, содержащей все возможные хеш-коды и значения, из которых они были получены.

Радужные таблицы сгенерированы для алгоритмов MD5 и SHA1, так что, в принципе, существует вероятность того, что хакеру удастся сравнительно легко вскрыть ваш пароль, если только вы не предпримете дополнительные меры безопасности.

#### Использование хеш-кодов с затравкой как средство усиления безопасности

Хотя это и не панацея, добавление затравки в алгоритм хеширования способно значительно затруднить хакерам взлом пользовательских паролей. Затравка — это строка, заранее заданная или выбираемая случайным образом, которая при хешировании добавляется к пользовательскому вводу.

Без использования затравки процедура хеширования пароля выглядит примерно так:

```
$hash = sha1($password);
```

Чтобы добавить затравку, можно применить, например, следующий код.

[1] `http://ru.wikipedia.org/wiki/Радужная_таблица`.

```
$salt = substr(md5(time()), 0, 7); // создаем случайную затравку
$hash = $salt . sha1($salt . $password);
```

Этот код генерирует случайное семизначное затравочное значение. Затравка присоединяется к строке пароля до хеширования. Это означает, что даже в случае совпадения паролей двух пользователей их хешированные значения будут различными.

В то же время, для того чтобы воспроизвести данный хеш-код, надо располагать соответствующей затравкой. Поэтому к хеш-коду присоединяется также и затравка, причем в незашифрованном виде. Благодаря этому, когда пользователь выполняет вход в систему, вы можете получить затравку из хеш-кода пароля, извлеченного из базы данных, и использовать ее для воссоздания хеш-кода с затравкой по паролю, введенному пользователем.

```
$salt = substr($dbhash, 0, 7); // извлечь затравку из сохраненного
                               // хеш-кода пароля
$hash = $salt . sha1($salt . $_POST['password']);
if ( $dbhash==$hash )
{
echo "Совпадает!";
}
else
{
echo "Не совпадает.";
```

**Хеш-коды с затравкой и радужные таблицы**

Добавление затравки делает радужные таблицы, по сути, бесполезными. В этом случае для вскрытия паролей пользователей потребуется сгенерировать новую таблицу, которая учитывала бы затравку. И хотя в принципе это вполне осуществимо, хакеру придется затратить на атаку значительно больше времени, а вы получите дополнительную линию защиты.

В большинстве приложений (особенно тех, которые не связаны с доступом к важной личной информации, например к данным кредитной карточки) использование паролей с затравкой является серьезным сдерживающим фактором в случае хакерских атак.

В качестве дополнительной меры безопасности рекомендуется ограничивать допустимое количество повторных неудачных попыток регистрации. Благодаря этому в распоряжении хакера будет всего лишь конечное количество попыток взломать пароль, поскольку все дальнейшие попытки будут блокироваться. Кроме того, это позволит противостоять атакам DOS (Denial of Service — отказ в обслуживании), т.е. атакам, целью которых является перегрузка сайта и вывод его из строя путем отправки огромного количества непрерывно следующих друг за другом запросов.

Создать такую функцию сравнительно несложно, и она должна решать следующие задачи.

1. Проверить, была ли предоставлена затравка. Если затравка не была передана, сгенерировать для нее новое значение путем хеширования временной метки UNIX, а затем извлечь из возвращенного результата подстроку, длина которой определяется значением переменной `$_saltLength`, и сохранить ее в переменной `$salt`.
2. Если затравка была передана, извлечь из хеш-кода с затравкой, взятого из базы данных, подстроку, длина которой определяется значением переменной `$_saltLength`, и сохранить ее в переменной `$salt`.
3. Присоединить затравку к хеш-коду объединенных значений затравки и пароля и возвратить новую строку.

Для выполнения указанных действий вставьте в класс `Admin` следующий метод.

```
<?php

class Admin extends DB_Connect
{

    private $_saltLength = 7;

    public function __construct($db=NULL, $saltLength=NULL) {...}

    public function processLoginForm() {...}

    /**
     * Генерирует хеш-код с затравкой для предоставленной строки
     *
     * @param string $string: подлежащая хешированию строка
     * @param string $salt: отсюда извлечь затравку
     * @return string: хеш-код с затравкой
     */
    private function _getSaltedHash($string, $salt=NULL)
    {
        /*
         * Сгенерировать затравку, если она не была предоставлена
         */
        if ( $salt==NULL )
        {
            $salt = substr(md5(time()), 0, $this->_saltLength);
        }

        /*
         * Извлечь затравку из строки, если она была передана
         */
        else
        {
            $salt = substr($salt, 0, $this->_saltLength);
        }

        /*
         * Добавить затравку в хеш-код и возвратить его
         */
        return $salt . sha1($salt . $string);
    }

}

?>
```

### Метод для тестирования хеш-кодов с затравкой

Чтобы продемонстрировать, как работает хеширование с использованием затравки, создадим для метода `_getSaltedHash()` простой тестовый метод под названием `testtSaltedHash()`. Это будет общедоступная функция, которая осуществляет требуемый вызов и выводит результирующие значения, что позволит увидеть, как функционирует сценарий.

Определите в классе `Admin` метод `testtSaltedHash()`.

```
<?php

class Admin extends DB_Connect
{
    private $_saltLength = 7;

    public function __construct($db=NULL, $saltLength=NULL) {...}

    public function processLoginForm() {...}

    private function _getSaltedHash($string, $salt=NULL) {...}

    public function testSaltedHash($string, $salt=NULL)
    {
        return $this->_getSaltedHash($string, $salt);
    }
}

?>
```

После этого добавьте новый файл `test.php`, который будет использовать функцию `testSaltedHash`, и поместите его в папку `public` (`/public/test.php`). Вызовите в этой функции файл инициализации, создайте новый класс `Admin` и выведите для слова *test* три значения хеш-кода Создайте первый хеш-код без использования затравки, затем выждите одну секунду для получения новой временной отметки. Создайте второй хеш-код без использования затравки и вновь выждите одну секунду. Наконец, создайте третий хеш-код, используя затравку из второго хеш-кода. Для выполнения этого теста вставьте в указанный файл следующий код.

```
<?php

// Включить необходимые файлы
include_once '../sys/core/init.inc.php';

// Загрузить объект Admin
$obj = new Admin($dbo);

// Загрузить хеш-код слова "test" и вывести его
$hash1 = $obj->testSaltedHash("test");
echo "{Хеш 1 без задания затравки:<br />", $hash1, "<br /><br />";

// Приостановить выполнение на 1 секунду для получения новой
// временной отметки
sleep(1);

// Загрузить второй хеш-код слова "test"
$hash2 = $obj->testSaltedHash("test");
echo " Хеш 2 без задания затравки:<br />", $hash2, "<br /><br />";

// Приостановить выполнение на 1 секунду для получения новой временной отметки
sleep(1);

// Заново хешировать слово "test" с использованием существующей затравки
$hash3 = $obj->testSaltedHash("test", $hash2);
echo " Хеш 3 с затравкой из хеша 2:<br />", $hash3;

?>
```

**Примечание.** Функция `sleep()` создает задержку при выполнении сценария, длительность которой (в секундах) задается единственным параметром этой функции.

Хотя полученные вами результаты будут отличаться от приведенных ниже из-за различий в хеш-кодах меток времени, используемых в качестве затравки, в целом они должны выглядеть примерно так.

```
Хеш 1 без задания затравки:
0286de19e9003b60d7b82686e94e84464eadb2b9b737884

Хеш 2 без задания затравки:
967ccec3d63bbff88dc263032fe9133d8b7d0f93ea74f75

Хеш 3 с затравкой из хеша 2:
967ccec3d63bbff88dc263032fe9133d8b7d0f93ea74f75
```

Как нетрудно заметить, независимо полученные значения хеш-кода для слова *test* не совпадают между собой, тогда как использование существующей затравки приводит к получению того же значения хеш-кода. Таким образом, даже если у двух пользователей имеются одинаковые пароли, их сохраненные хеш-коды будут различными, что значительно затруднит вскрытие паролей для потенциальных злоумышленников.

## Создание учетной записи пользователя для тестирования административного доступа

Чтобы можно было протестировать функции администрирования, в таблице `users` должна существовать пара значений "имя_пользователя/пароль". Для простоты выберем в качестве имени пользователя `testuser`, в качестве пароля — `admin`, а в качестве адреса электронной почты — `admin@example.com`. Имейте в виду, что выбранный нами пароль *не является безопасным*; он используется здесь исключительно в демонстрационных целях, и при переходе к любому производственному сценарию его следует заменить.

Начнем с того, что сгенерируем хеш-код для пароля `admin`, что легко делается с помощью тестового метода `testSaltedHash` и файла `test.php`. Чтобы сгенерировать хеш-код с затравкой для тестового пароля пользователя, добавьте в файл `test.php` следующий код, как показано ниже.

```
<?php

// Включить необходимые файлы
include_once '../sys/core/init.inc.php';

// Загрузить объект Admin
$obj = new Admin($dbo);

// Сгенерировать хеш-код с затравкой для пароля "admin"
$pass = $obj->testSaltedHash("admin");
echo 'Хеш-код пароля "admin":<br />', $pass, "<br /><br />";

?>
```

Перейдя на страницу `http://localhost/test.php`, вы увидите примерно следующий результат.

```
Хеш-код пароля "admin":
8451e0938f510d4aad8975efa57f9f1d72d797dab986ef4
```

Скопируйте полученный хеш-код в буфер обмена, перейдите на страницу `http://localhost/phpmyadmin` и щелкните на вкладке SQL. Чтобы вставить данные пользователя `testuser` в таблицу, выполните следующий запрос.

```
INSERT INTO `php-jquery_example`.`users`
 (`user_name`, `user_pass`, `user_email`)
VALUES
 (
    'testuser',
    '8451e0938f510d4aad8975efa57f9f1d72d797dab986ef4',
    'admin@example.com'
 );
```

Выполнив предыдущий код, щелкните на базе данных `php-jquery_example`, а затем — на таблице `users`. Перейдите на вкладку Обзор (Browse) для просмотра содержащихся в таблице данных о пользователе (рис. 6.3).

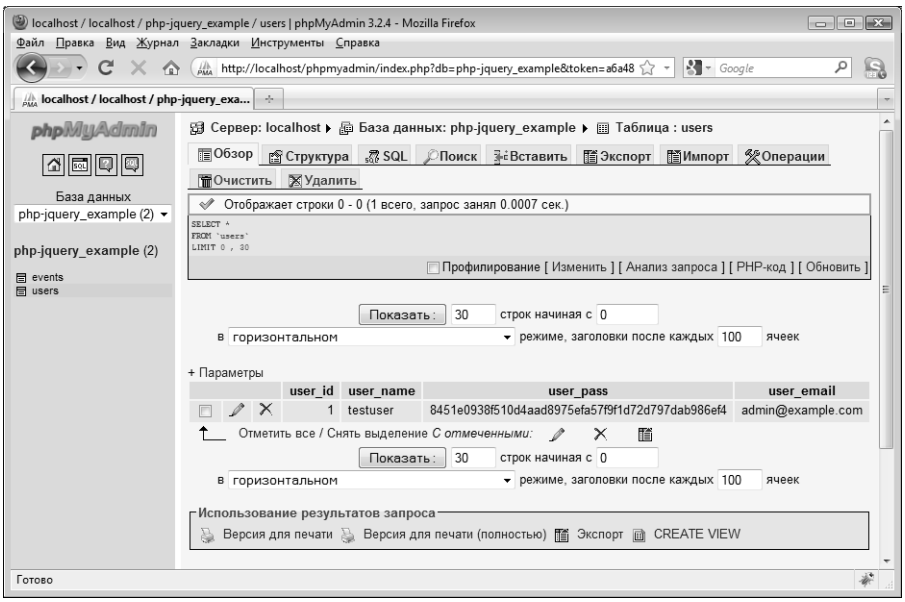



**Рис. 6.3.** Отображение информации о пользователе `testuser` после ее внесения в базу данных

После внесения в базу данных информации о пользователе, зашифрованной с использованием хеш-кода с затравкой, можно удалить метод `testSaltedHash()` из класса `Admin` и целиком удалить файл `test.php`.

## Модификация приложения для обработки отправки регистрационной формы

На данном этапе мы уже почти готовы приступить к тестированию созданной учетной записи пользователя, но остается еще внести в файл `process.inc.php` изменения, обеспечивающие отправку данных регистрационной формы.

Все, что для этого надо сделать, — это добавить новый элемент в массив `$actions`. Откройте файл `process.inc.php` и вставьте в него дополнительный код, выделенный ниже полужирным шрифтом.

```
<?php

/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_edit' => array(
                'object' => 'Calendar',
                'method' => 'processForm',
                'header' => 'Location: ../../'
            ),
        'user_login' => array(
                'object' => 'Admin',
                'method' => 'processLoginForm',
                'header' => 'Location: ../../'
            )
    );

/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( $_POST['token']==$_SESSION['token']
     && isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);
    if ( TRUE === $msg=$obj->$use_array['method']() )
    {
        header($use_array['header']);
        exit;
    }
    else
```

```
    {
        // В случае ошибки вывести сообщение о ней и прекратить выполнение
        die ( $msg );
    }
}
else
{
    // В случае некорректности маркера/действия перенаправить
    // пользователя на основную страницу
    header("Location: ../../");
    exit;
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}

?>
```

Теперь можно протестировать созданную учетную запись. Поскольку никакие проверки учетной записи нами пока не предусмотрены, просто добавим в файл `index.php` условный оператор, позволяющий отобразить текущее состояние учетной записи, вставив для этого дополнительный код, выделенный ниже полужирным шрифтом.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь для января
 */
 $cal = new Calendar($dbo, "2010-01-01 12:00:00");

/*
 * Задать название страницы и файлы CSS
 */
$page_title = "Календарь событий";
$css_files = array('style.css', 'admin.css');

/*
 * Включить начальную часть страницы
 */
include_once 'assets/common/header.inc.php';

?>
```

```
<div id="content">

<?php

/*
 * Отобразить календарь в виде HTML
 */
echo $cal->buildCalendar();

?>

</div><!-- end #content -->
<p>
<?php

    echo isset($_SESSION['user']) ? " Вход выполнен!" :
         " Вход не выполнен!";

?>
</p>

<?php

/*
 * Включить завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохраните файл и перейдите на страницу `http://localhost/`, где можно будет увидеть, что непосредственно под календарем отображается сообщение *Вход не выполнен!* (рис. 6.4).

После этого перейдите на страницу `http://localhost/login.php` и введите имя пользователя `testuser` и пароль `admin` (рис. 6.5).

После щелчка на кнопке Вход отобразится основная страница календаря, но теперь отображаемое в нижней части календаря сообщение будет другим: *Вход выполнен!* (рис. 6.6).

# Предоставление зарегистрированному пользователю возможности завершения сеанса

Далее мы добавим метод, предоставляющий зарегистрированному пользователю возможность завершить сеанс. Это будет сделано с помощью формы, отправляющей информацию файлу `process.inc.php`. Для генерации этой формы используем метод `_adminGeneralOptions()` класса `Calendar`.

## Добавление кнопки выхода

Чтобы добавить кнопку, которую зарегистрированные пользователи могли бы использовать для завершения сеанса, видоизменим метод `_adminGeneralOptions` класса `Calendar`. Теперь, кроме предоставления кнопки для добавления новых событий, этот метод будет выводить форму, отправляющую файлу `process.inc.php` маркер сеанса и идентификатор вида действия `user_logout`. Откройте класс `Calendar`

и внесите в метод `_adminGeneralOptions()` изменения, выделенные ниже полужирным шрифтом.

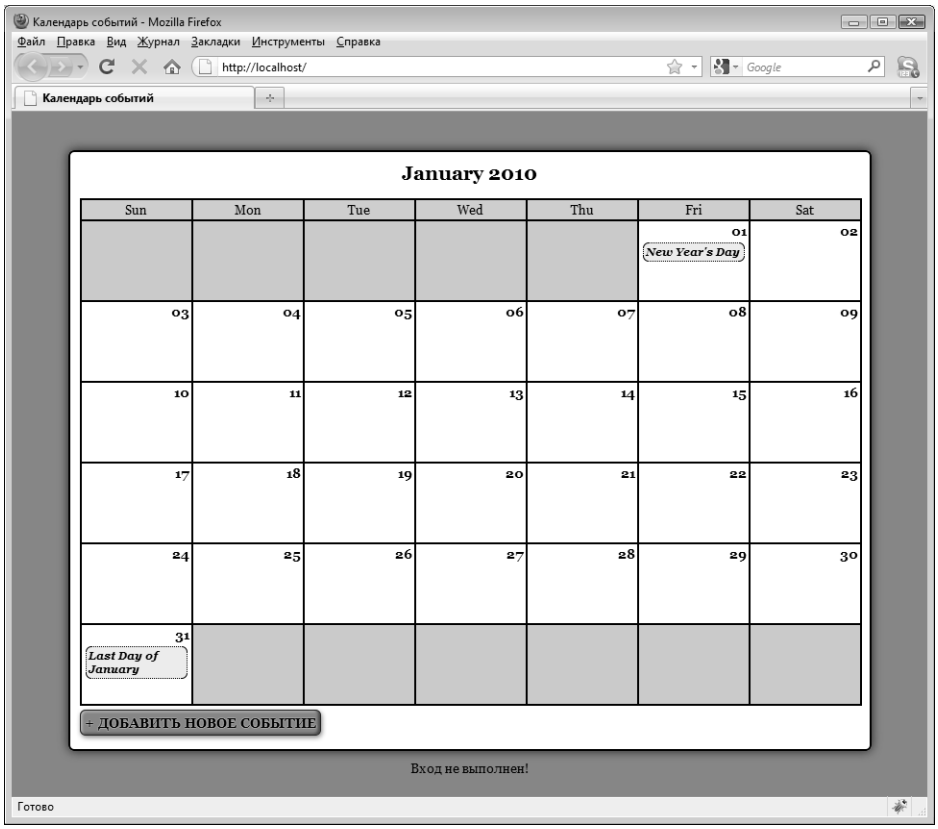



**Рис. 6.4.** До выполнения пользователем процедуры входа под календарем отображается сообщение "Вход не выполнен!"

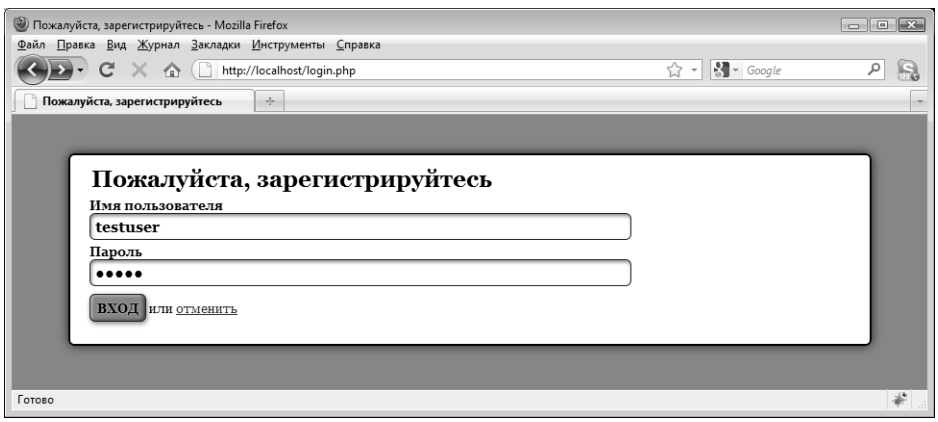



**Рис. 6.5.** Регистрационная форма, в которой введены имя пользователя и пароль

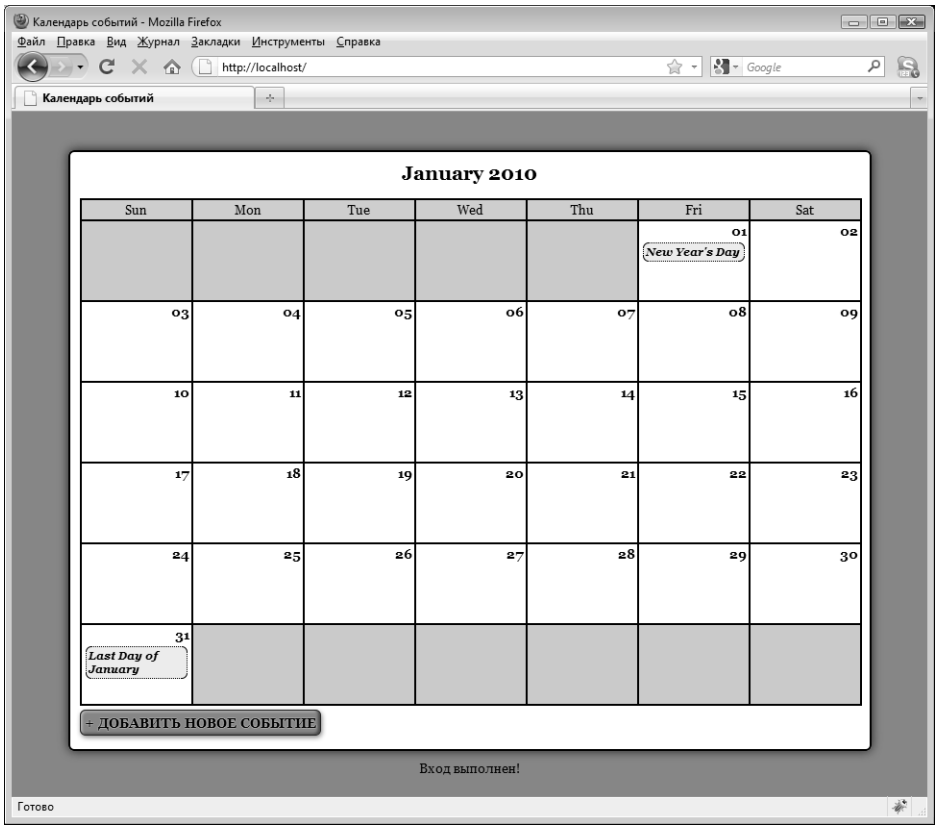



**Рис. 6.6.** После выполнения пользователем процедуры входа в нижней части календаря отображается сообщение “Вход выполнен!”

```
    private function _adminGeneralOptions()
    {
        /*
         * Отобразить административные элементы управления
         */
        return <<<ADMIN_OPTIONS

    <a href="admin.php" class="admin">+ Добавить новое событие</a>
    <form action="assets/inc/process.inc.php" method="post">
        <div>
            <input type="submit" value="Выход" class="admin" />
            <input type="hidden" name="token"
                value="$_SESSION[token]" />
            <input type="hidden" name="action"
                value="user_logout" />
        </div>
    </form>
ADMIN_OPTIONS;
    }
```

Сохранив изменения и обновив страницу `http://localhost/`, вы увидите, что в календаре появилась новая кнопка (рис. 6.7).

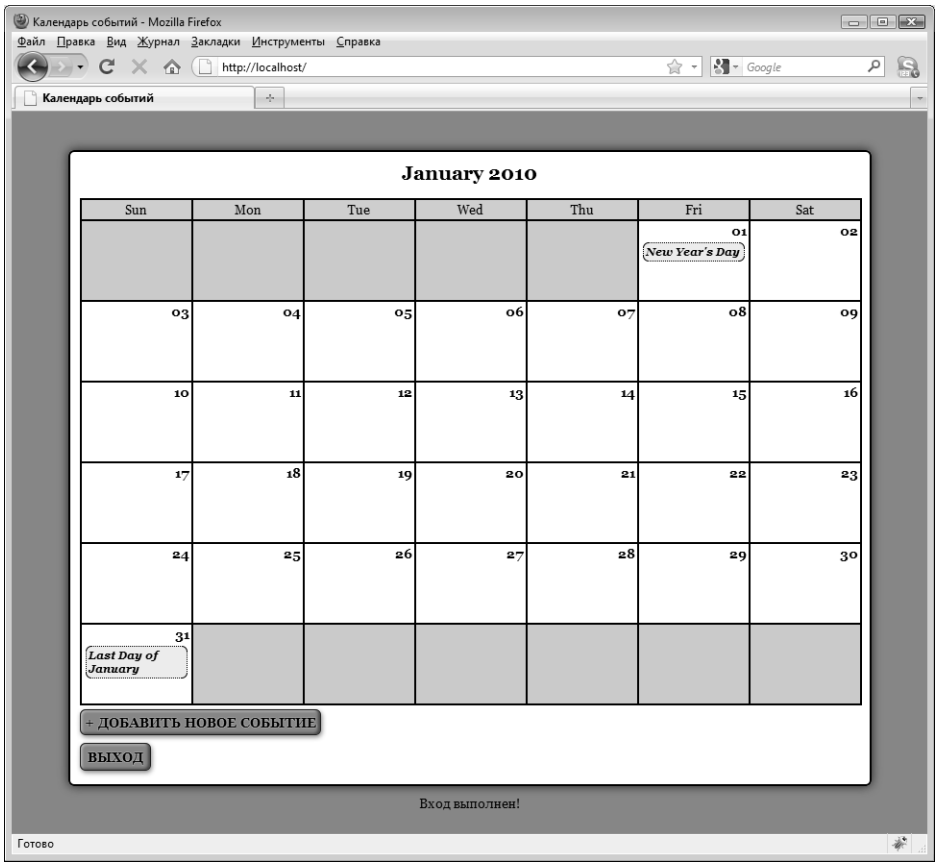



**Рис. 6.7.** Отображение кнопки Выход после внесения изменений в класс `Calendar`

## Создание метода для процедуры выхода

Чтобы обеспечить выполнение процедуры выхода, добавим в класс `Admin` новый общедоступный метод `processLogout()`. Данный метод будет осуществлять проверку того, что отправленное значение идентификатора вида действия совпадает с требуемым (`user_logout`), а затем — использовать функцию `session_destroy()` для удаления массива данных о пользователе путем полного уничтожения текущего сеанса.

Добавим этот метод в класс `Admin`, введя следующий дополнительный код, выделенный полужирным шрифтом.

```
<?php

class Admin extends DB_Connect
{

    private $_saltLength = 7;
```

```
    public function __construct($db=NULL, $saltLength=NULL) {...}

    public function processLoginForm() {...}

    /**
     * Завершает сеанс пользователя
     *
     * @return mixed: TRUE в случае успешного завершения, иначе --
     *                сообщение об ошибке
     */
    public function processLogout()
    {
        /*
         * Аварийное завершение, если было передано неправильное
         * значение атрибута ACTION
         */
        if ( $_POST['action']!='user_logout' )
        {
            return "В processLogout было передано неправильное
                    значение атрибута ACTION.";
        }

        /*
         * Удалить массив user из текущего сеанса
         */
        session_destroy();
        return TRUE;
    }

    private function _getSaltedHash($string, $salt=NULL) {...}

}

?>
```

## Модификация приложения для обработки завершения сеанса

Последнее, что нам остается сделать для того, чтобы пользователь мог успешно завершить сеанс, — это добавить еще один элемент в массив `$actions` в файле `process.inc.php`. Для этого вставьте в указанный файл дополнительный код, выделенный ниже полужирным шрифтом.

```
<?php

/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
```

```
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_edit' => array(
                'object' => 'Calendar',
                'method' => 'processForm',
                'header' => 'Location: ../../'
            ),
        'user_login' => array(
                'object' => 'Admin',
                'method' => 'processLoginForm',
                'header' => 'Location: ../../'
            ),
        'user_logout' => array(
                'object' => 'Admin',
                'method' => 'processLogout',
                'header' => 'Location: ../../'
            )
    );

/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( $_POST['token']==$_SESSION['token']
        && isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);
    if ( TRUE === $msg=$obj->$use_array['method']() )
    {
        header($use_array['header']);
        exit;
    }
    else
    {
        // В случае ошибки вывести сообщение о ней и
        // прекратить выполнение
        die ( $msg );
    }
}
else
{
```

```
    // В случае некорректности маркера/действия перенаправить
    // пользователя на основную страницу
    header("Location: ../../");
    exit;
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}

?>
```

Сохраните этот файл, перезагрузите страницу `http://localhost/` и щелкните на кнопке Выход в нижней части календаря. Текст сообщения, отображаемого под календарем, изменится и станет таким: *Вход не выполнен!* (рис. 6.8).

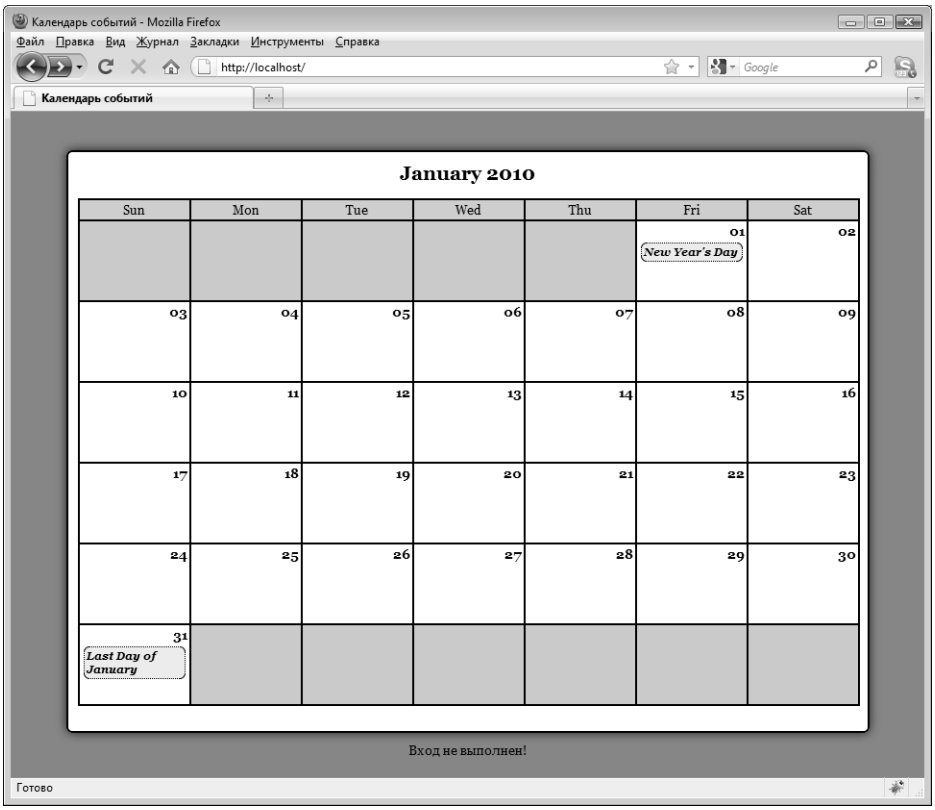



**Рис. 6.8.** Щелчок на кнопке Выход удаляет из сеанса информацию о пользователе

**Примечание.** Теперь, когда вы уже знаете, как работает процедура входа, можете удалить из файла `index.php` блок кода, ответственного за вывод сообщений *`Вход выполнен!`* и *`Вход не выполнен!`*, вместе с окружающими его дескрипторами абзаца `<p>`.

# Отображение элементов административного управления

После того как мы организовали в приложении выполнение процедур входа и выхода, последнее, что еще остается сделать, — это предусмотреть, чтобы любые операции и опции, требующие административного доступа, отображались лишь для зарегистрированных пользователей, т.е. тех, которые выполнили процедуру входа.

## Отображение административных опций лишь для администраторов

Кнопки, позволяющие добавлять и редактировать события, должны отображаться только в том случае, если пользователь выполнил процедуру входа. Для организации соответствующей проверки необходимо модифицировать методы `_adminGeneralOptions()` и `_adminEntryOptions()` класса `Calendar`.

### *Модификация метода `_adminGeneralOptions()`*

Обратимся к общим опциям календаря. Если пользователь выполнил вход в систему, то ему необходимо предоставить опцию создания нового события, а также опцию выхода.

Вместе с тем, если пользователь не выполнил процедуру входа, ему должна быть доступна ссылка, с помощью которой он мог бы это сделать. Для обеспечения соответствующих проверок внесите в метод `_adminGeneralOptions()` класса `Calendar` изменения, выделенные в приведенном ниже листинге полужирным шрифтом.

```
<?php

    class Calendar extends DB_Connect
    {

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    public function buildCalendar() {...}

    public function displayForm() {...}
```

```
    public function processForm() {...}

    public function confirmDelete($id) {...}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

    private function _adminGeneralOptions()
    {
        /*
         * Если пользователь выполнил процедуру входа,
         * отобразить средства администрирования
         */
        if ( isset($_SESSION['user']) )
        {
            return <<<ADMIN_OPTIONS

    <a href="admin.php" class="admin">+ Добавить новое событие</a>
    <form action="assets/inc/process.inc.php" method="post">
        <div>
            <input type="submit" value="Выход" class="admin" />
            <input type="hidden" name="token"
                value="$_SESSION[token]" />
            <input type="hidden" name="action"
                value="user_logout" />
        </div>
    </form>
ADMIN_OPTIONS;
        }
        else
        {
            return <<<ADMIN_OPTIONS

    <a href="login.php">Log In</a>
ADMIN_OPTIONS;
        }
    }

    private function _adminEntryOptions($id) {...}

}

?>
```

Сохраните изменения и перезагрузите страницу `http://localhost/` в состоянии, когда вход не выполнен. Вы увидите, что административные элементы управления исчезли и вместо них появилась простая ссылка *Регистрация* (рис. 6.9).

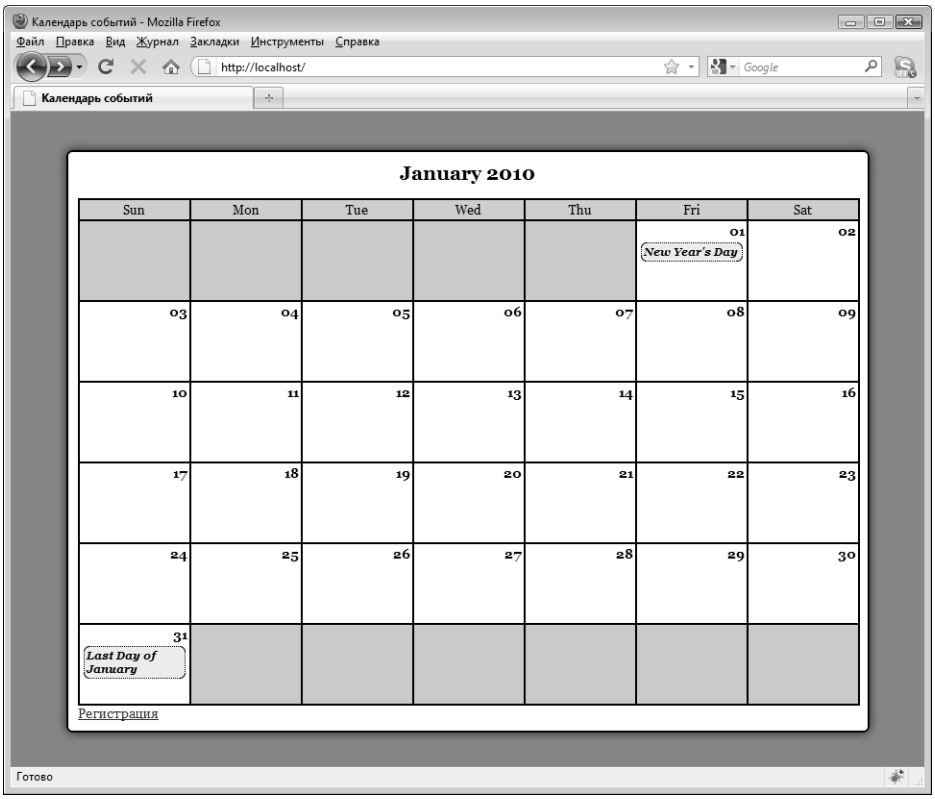



**Рис. 6.9.** Если пользователь не выполнил процедуры входа в систему, для него отображается только ссылка `Регистрация`

## Модификация метода `_adminEventOptions()`

Далее мы добавим код, исключающий возможность редактирования и удаления событий незарегистрированными пользователями. Для этого изменим метод `_adminEventOptions()` класса `Calendar`, как показано ниже.

```
<?php

class Calendar extends DB_Connect
{
    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}
```

```
    public function buildCalendar() {...}

    public function displayForm() {...}

    public function processForm() {...}

    public function confirmDelete($id) {...}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

    private function _adminGeneralOptions() {...}

    private function _adminEntryOptions($id)
    {
        if ( isset($_SESSION['user']) )
        {
            return <<<ADMIN_OPTIONS

    <div class="admin-options">
    <form action="admin.php" method="post">

        <p>
            <input type="submit" name="edit_event"
                  value="Редактировать событие" />
            <input type="hidden" name="event_id"
                  value="$id" />
        </p>
    </form>
    <form action="confirmdelete.php" method="post">
        <p>
            <input type="submit" name="delete_event"
                  value="Удалить событие" />
            <input type="hidden" name="event_id"
                  value="$id" />
        </p>
    </form>
    </div><!-- end .admin-options -->
ADMIN_OPTIONS;
        }
        else
        {
            return NULL;
        }
    }

}

?>
```

Сохраните изменения и перезагрузите страницу `http://localhost/` в состоянии, когда вход не выполнен. Щелкните на любом событии для просмотра его полного описания, и вы увидите, что средства административного управления уже не отображаются (рис. 6.10).

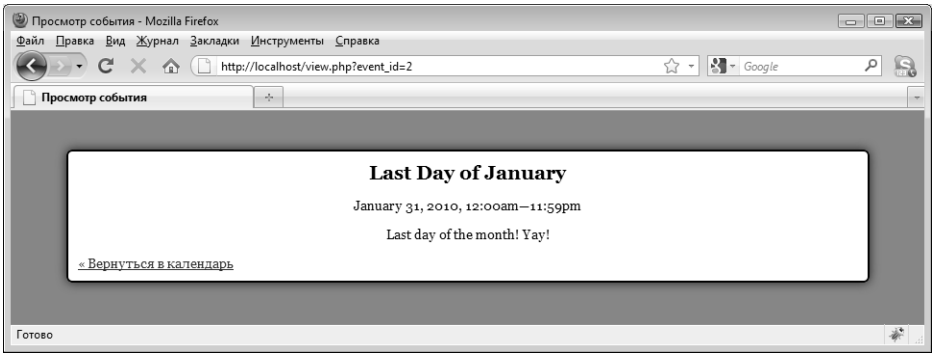



**Рис. 6.10.** Представление подробной информации о событии в состоянии, когда пользователь не выполнил процедуры входа в систему

## Ограничение доступа к административным страницам

В качестве дополнительной меры безопасности организуем предварительную проверку полномочий пользователя, прежде чем предоставлять ему возможность доступа к таким, например, страницам, как форма для создания и редактирования событий.

### *Предотвращение создания событий незарегистрированными пользователями*

Чтобы пресечь попытки незарегистрированных пользователей получить доступ к форме для создания новых событий, организуем простую проверку, внеся изменения в соответствующий файл. Если пользователь, не выполнивший процедуру входа, попытается запустить сценарий, то эта попытка окажется неудачной, и пользователь будет автоматически перенаправлен на основную страницу календаря.

Для реализации этого откройте файл `admin.php` и добавьте в него код, выделенный в приведенном ниже листинге полужирным шрифтом.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Перенаправить незарегистрированного пользователя на
 * основную страницу
 */
if ( !isset($_SESSION['user']) )
{
```

```
header("Location: ./");
exit;
}

/*
 * Вывести начальную часть страницы
 */
$page_title = "Добавление/редактирование события";
$css_files = array("style.css", "admin.css");
include_once 'assets/common/header.inc.php';

/*
* Загрузить календарь
*/
$cal = new Calendar($dbo);

?>

<div id="content">
<?php echo $cal->displayForm(); ?>

</div><!-- end #content -->

<?php

/*
* Вывести завершающую часть страницы
*/
include_once 'assets/common/footer.inc.php';

?>
```

Сохранив файл, попытайтесь перейти в браузере на страницу `http://localhost/admin.php` в состоянии, когда вход не выполнен. Вы будете автоматически перенаправлены на основную страницу `http://localhost/`.

### Предоставление возможности удаления событий лишь зарегистрированным пользователям

В дополнение к предпринятым выше мерам безопасности, исключим возможность удаления событий незарегистрированными пользователями, введя в файл `confirmdelete.php` изменения, выделенные в приведенном ниже листинге полужирным шрифтом.

```
<?php

/*
 * Начать сеанс
 */
session_start();

/*
 * Убедиться в том, что ID был передан и пользователь выполнил вход
 */
if ( isset($_POST['event_id']) && isset($_SESSION['user']) )
```

```
{
    /*
     * Извлечь ID события из строки URL
     */
    $id = (int) $_POST['event_id'];
}
else
{
    /*
     * Если не был предоставлен ID или пользователь не выполнил
     * процедуру входа, принудительно перейти на основную страницу
     */
    header("Location: ./");
    exit;
}

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь
 */
$cal = new Calendar($dbo);
$markup = $cal->confirmDelete($id);

/*
 * Вывести начальную часть страницы
 */
$page_title = "Просмотр события";
$css_files = array("style.css", "admin.css");
include_once 'assets/common/header.inc.php';

?>

<div id="content">
<?php echo $markup; ?>

</div><!-- end #content -->

<?php

/*
 * Вывести завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

Сохраните этот код и попытайтесь получить доступ к странице `http://localhost/confirmdelete.php` в состоянии, когда вход не выполнен. Как и следовало ожидать, вы будете перенаправлены на страницу `http://localhost/`.

# Резюме

В этой главе вы узнали о том, как организовать в приложении процедуру авторизации пользователей, т.е. сделать так, чтобы вносить изменения в календарь могли только пользователи, имеющие на это полномочия. Вы научились создавать предназначенные для этого классы (`Admin`), организовывать проверку учетных данных пользователя, отображать средства администрирования только для администраторов и ограничивать доступ к административным страницам. В следующей главе вы приступите к внедрению средств jQuery в приложение, последовательно делая его все более удобным в использовании.

# Часть III

# Добавление сценариев jQuery в PHP-приложения

Сейчас, когда календарь нормально функционирует, мы можем задействовать jQuery для улучшения пользовательского интерфейса приложения. В следующих главах мы добавим в календарь функциональность AJAX, расширив приложение сценариями JavaScript.

# Глава 7

# Улучшение пользовательского интерфейса средствами jQuery

Уже на данном этапе наше приложение является полностью работоспособным. Оно позволяет просматривать события, а пользователи с административными правами могут регистрироваться в нем для создания, редактирования и удаления событий.

Наша следующая задача — окончательная шлифовка приложения для создания завершенного пользовательского интерфейса путем добавления функциональных возможностей AJAX с соблюдением принципов так называемого прогрессивного улучшения.

## Прогрессивное улучшение приложения с помощью jQuery

*Прогрессивное улучшение* (progressive enhancement) — это термин, впервые использованный Стивеном Чампеоном (Steven Champeon)[1] для описания нового подхода к разработке веб-приложений, при котором они проектируются так, чтобы быть доступными для любых интернет-подключений и браузеров любого типа за счет использования семантического HTML и других технологий, применяемых послойно (например, CSS-файлы или разметка JavaScript)[2].

Чтобы приложение укладывалось в русло идеологии прогрессивного улучшения, его разработка должна вестись в соответствии со следующими принципами.

- Базовое содержимое должно быть доступным для браузеров любого типа, что достигается за счет использования простейшей, максимально семантизированной HTML-разметки.
- Базовая функциональность приложения обеспечивается во всех браузерах.

[1] `http://www.hesketh.com/about-us/leadership-team`.

[2] Своеобразным "антиподом" и вместе с тем аналогом термина *прогрессивное улучшение* (progressive enhancement) является термин *постепенное сокращение возможностей* (graceful degradation). Оба термина предполагают реализацию приложений с использованием наиболее совершенных из имеющихся на сегодняшний день средств и эффектов, но таким образом, чтобы это не препятствовало просмотру страниц в браузерах с ограниченными функциональными возможностями. — *Примеч. ред.*

- Приоритет отдается пользовательским предпочтениям, т.е. приложение не должно изменять параметры (например, размер окна), заданные для браузера пользователем.
- Представление и стилевое оформление документа обеспечиваются CSS-файлами, присоединяемыми с помощью внешних ссылок.
- Пользовательский интерфейс улучшается за счет средств JavaScript, присоединяемых с помощью внешних ссылок, при этом сценарии JavaScript должны оставаться *ненавязчивыми*[3] и не играть существенной роли в обеспечении нормальной работы приложения.

Наше приложение уже соответствует первым четырем критериям (оно будет работать даже при отключенных стилях, хотя при этом его внешний вид и потеряет привлекательность).

Таким образом, коль скоро средства JavaScript не будут использованы нами для создания новой функциональности приложения в ходе его дальнейшей доработки, мы будем успешно следовать принципам прогрессивного улучшения.

## Постановка задачи

Действуя в соответствии с принципами прогрессивного улучшения, мы добавим в приложение возможность просмотра информации о событиях в *модальном окне*, т.е. в области содержимого, располагающейся поверх существующей разметки, без обновления страницы. Обычно такие окна запускаются средствами JavaScript и используются на многих современных веб-сайтах.

В календаре модальные окна будут применены для отображения подробной информации о событии после щелчка на его названии. Это будет делаться без обновления страницы, за счет использования AJAX.

# Подключение jQuery к приложению

Как вам уже должно быть известно, для того чтобы использовать синтаксис jQuery, требуется сначала подключить данную библиотеку к приложению. Поскольку файлы JavaScript должны располагаться в самом конце HTML-разметки перед закрывающим дескриптором тела документа (`</body>`), используйте для включения библиотеки jQuery и других файлов, которые могут потребоваться на этом этапе, файл `footer.inc.php` (`/public/assets/common/footer.inc.php`). Начните с включения самой последней версии jQuery в приложение; для этого добавьте в файл `footer.inc.php` следующий код, выделенный полужирным шрифтом.

```
    <script type="text/javascript"
        src="http://www.google.com/jsapi"></script>
    <script type="text/javascript">
        google.load("jquery", "1");
    </script>
</body>

</html>
```

[3] Сутью принципа *ненавязчивого JavaScript* (unobtrusive JavaScript) является отделение функциональности веб-страницы от ее представления (см., например, `http://ru.wikipedia.org/wiki/Ненавязчивый_JavaScript`). — *Примеч. ред.*

Сохраните код и загрузите в браузер страницу `http://localhost`. Откройте консоль Firebug и выполните приведенную ниже команду, чтобы убедиться в успешности загрузки jQuery в приложение.

```
$("h2").text();
```

В консоли должен отобразиться следующий результат.

```
>>> $("h2").text();
"January 2010"
```

**Примечание.** Поскольку мы используем Google JSAPI, кроме сервера Apache вам потребуется еще и доступ к Интернету. Если у вас отсутствует такой доступ или вы предпочитаете не использовать его, можете вместо этого загрузить последнюю версию jQuery с сайта `http://jquery.com` и включить ее в приложение.

## Создание файла инициализации JavaScript

Наше приложение должно подчиняться принципам прогрессивного улучшения, поэтому все сценарии будут помещены во внешний файл `init.js`. Он будет находиться в общедоступной папке `js` (`/public/assets/js/init.js`) и содержать весь пользовательский код jQuery для приложения.

### Подключение файла инициализации к приложению

Прежде чем ваши сценарии станут доступными для приложения, в него необходимо включить файл инициализации. Приложение будет использовать синтаксис jQuery, поэтому файл инициализации следует включить после сценария, который загружает jQuery в файле `footer.inc.php`.

```
    <script type="text/javascript"
        src="http://www.google.com/jsapi"></script>
    <script type="text/javascript">
        google.load("jquery", "1");
    </script>
    <script type="text/javascript"
          src="assets/js/init.js"></script>

</body>

</html>
```

### Проверка готовности документа перед выполнением сценария

Создав файл `init.js`, прежде всего позаботьтесь о том, чтобы сценарии JavaScript могли выполняться лишь после того, как объектная модель документа будет готова к использованию. Для этого воспользуемся предусмотренной в jQuery сокращенной записью вызова `$(document).ready()` и при этом дополнительно повысим отказоустойчивость кода, явно указав псевдоним `$` для функции `jQuery()` в качестве аргумента анонимной функции обратного вызова, вместо того чтобы использовать его как глобально определенный псевдоним. Вставьте в файл `init.js` следующий код.

```
// Удостовериться в готовности документа, прежде чем выполнять сценарии
jQuery(function($){

// Быстрая проверка того, что сценарий действительно загрузился
console.log("Файл init.js успешно загружен.");

});
```

Сохраните файл и загрузите страницу `http://localhost/` в браузере с открытой консолью Firebug. После того как файл загрузится, в консоли отобразится следующее сообщение.

```
Файл init.js успешно загружен.
```

## Создание новой таблицы стилей для элементов, созданных jQuery

Чтобы создаваемые впоследствии элементы jQuery отображались в требуемом виде, мы немного опередим события и подготовим еще один CSS-файл, предназначенный для хранения информации о стилях для элементов, которые вскоре будут создаваться сценариями jQuery.

Назовем этот файл `ajax.css` и поместим его в папку `css` (`/public/assets/css/ajax.css`). Создав файл, введите в него следующие описания стилей.

```
.modal-overlay {
    position: fixed;
    top: 0;
    left: 0;
    bottom: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0,0,0,.5);
    z-index: 4;
}

.modal-window {
    position: absolute;
    top: 140px;
    left: 50%;
    width: 300px;
    height: auto;
    margin-left: -150px;
    padding: 20px;
    border: 2px solid #000;
    background-color: #FFF;
    -moz-border-radius: 6px;
    -webkit-border-radius: 6px;
    border-radius: 6px;
    -moz-box-shadow: 0 0 14px #123;
    -webkit-box-shadow: 0 0 14px #123;
    box-shadow: 0 0 14px #123;
    z-index: 5;
}

    .modal-close-btn {
    position: absolute;
    top: 0;
    right: 4px;
    margin: 0;
    padding: 0;
    text-decoration: none;
    color: black;
```

```
    font-size: 16px;
}

    .modal-close-btn:before {
    position: relative;
    top: -1px;
    content: "Close";
    text-transform: uppercase;
    font-size: 10px;
}
```

### Включение таблицы стилей в файл `index.php`

Откройте файл `index.php` и включите новую таблицу стилей в массив `$css_files`, добавив строку, выделенную ниже полужирным шрифтом.

```
<?php

/*
 * Включить необходимые файлы
 */
include_once '../sys/core/init.inc.php';

/*
 * Загрузить календарь для января
 */
 $cal = new Calendar($dbo, "2010-01-01 12:00:00");

/*
 * Задать название страницы и файлы CSS
 */
$page_title = "Календарь событий";
$css_files = array('style.css', 'admin.css', 'ajax.css');

/*
 * Включить начальную часть страницы
 */
include_once 'assets/common/header.inc.php';

?>

<div id="content">

<?php

/*
 * Отобразить календарь в виде HTML
 */
echo $cal->buildCalendar();

?>

</div><!-- end #content -->

<?php

/*
```

```
 * Включить завершающую часть страницы
 */
include_once 'assets/common/footer.inc.php';

?>
```

# Создание модального окна для отображения информации о событии

Модальное окно для этого приложения будет довольно простым. Предназначенная для его создания процедура будет выглядеть следующим образом:

- предотвратить выполнение действия, предусмотренного по умолчанию (открытие представления `view.php`, отображающего подробное описание события);
- добавить класс `active` к ссылке на событие в календаре;
- извлечь строку запроса из атрибута `href` ссылки на событие;
- создать кнопку, щелчок на которой закрывает модальное окно;
- создать само модальное окно и поместить в него кнопку Закрыть;
- извлечь информацию из базы данных с помощью AJAX и отобразить ее в модальном окне.

Выполнение описанных действий происходит после запуска события `click`, инициируемого щелчком на ссылке названия календарного события.

## Связывание функции с событием щелчка на ссылке названия

Начнем с того, что добавим в файл `init.js` новый селектор, который выбирает все элементы `<a>`, являющиеся прямыми потомками элементов списка (`li>a`), и используем метод `live()` для привязки обработчика к событию `click`. Вставьте в файл `init.js` следующий код, выделенный полужирным шрифтом.

```
// Удостовериться в готовности документа, прежде чем выполнять сценарии
jQuery(function($){

// Захватывать события в модальном окне
$("li>a").live("click", function(event){

        // Поместить сюда код обработчика события

    });

});
```

## Предотвращение выполнения действия по умолчанию и добавление класса `active`

Далее необходимо предотвратить выполнение действия, предусмотренного по умолчанию, используя для этого метод `.preventDefault()`, а затем следует добавить класс `active` в элемент, на котором был выполнен щелчок, используя метод `.addClass()`.

С этой целью добавьте в файл `init.js` следующий код, выделенный полужирным шрифтом.

```
// Удостовериться в готовности документа, прежде чем выполнять сценарии
jQuery(function($){

// Захватывать события в модальном окне
$("li>a").live("click", function(event){

      // Предотвратить загрузку файла view.php по щелчку на ссылке
      event.preventDefault();

      // Добавить в ссылку класс "active"
      $(this).addClass("active");

      // Подтвердить тот факт, что сработал обработчик событий,
      // выводом текста ссылки
      console.log( $(this).text() );

    });

});
```

Сохраните изменения, перезагрузите страницу `http://localhost/` в браузере и щелкните на названии любого события. Перехода к файлу `view.php` для отображения информации о событии не произойдет; вместо этого в консоли отобразится название события. Например, если щелкнуть на событии *New Year's Day*, то в консоли отобразится следующий результат.

```
New Year's Day
```

## Извлечение строки запроса с помощью регулярных выражений

Модальное окно создается для отображения информации о событии, поэтому мы должны каким-то образом определить, какое именно событие следует отобразить. Не создавая никакой дополнительной разметки, можно извлечь идентификатор события непосредственно из ссылки `href` с помощью регулярных выражений.

Для этого необходимо извлечь из ссылки строку запроса. (Если значением атрибута `href` является `http://localhost/view.php?event_id=1`, то строкой запроса является `event_id=1`.)

При извлечении строк будут использоваться два элемента: собственный метод Java Script `.replace()` и регулярные выражения. Метод `.replace()` принимает строку или шаблон регулярного выражения, по которым осуществляется поиск, и строку или шаблон замены, которыми должны быть заменены найденные соответствия.

### *Простой подход: замена на основе строк*

На первый взгляд весьма соблазнительным кажется следующее простое решение:

```
var data = string.replace("http://localhost/view.php?", "");
```

Это действительно сработает, давая на выходе строку `"event_id=1"` (в предположении, что исходным значением `$string` является `http://localhost/view.php?event_id=1`). К сожалению, такому подходу не хватает гибкости. Что если по-

требуется переместить приложение в другой домен или же изменить имя файла на `event.php`? Любое из этих изменений нарушит предшествующую логику и потребует обновления сценария.

### Лучшее решение: регулярные выражения

В то же время существует гораздо более эффективное решение: *регулярные выражения*. Регулярные выражения — это мощнейший механизм поиска по шаблону, доступный в большинстве современных языков программирования. Чтобы извлечь строку запроса, используем шаблон, который осуществляет поиск первого вопросительного знака (`?`) в строке, а затем возвращает все, что расположено после него. Этот шаблон выглядит так:

```
/.*?\?(.*)$/
```

В качестве ограничителя в регулярных выражениях JavaScript используется косая черта (`/`), которая ставится в конце каждого выражения. Внутри данного выражения шаблон просматривает (в направлении слева направо) нуль или большее количество любых символов, пока не достигнет первого встретившегося вопросительного знака, а затем сохраняет все символы, следующие за вопросительным знаком до конца строки, в виде поименованной группы для использования при возможных последующих заменах.

---

**Примечание.** Более подробное рассмотрение регулярных выражений и способов их использования приведено в главе 9.

---

### Внедрение регулярных выражений в сценарий

Нам необходимо извлечь значение `href` ссылки, на которой был выполнен щелчок, поэтому воспользуемся ключевым словом `this`. Для применения методов jQuery необходимо сначала передать это ключевое слово в функцию jQuery. Тогда останется лишь получить доступ к значению `href` с помощью метода `.attrib()`, а затем вызвать метод `.replace()` и извлечь строку запроса.

При использовании регулярных выражений в методе `.replace()` нельзя окружать шаблон ни одинарными, ни двойными кавычками. Действуя, как только что было описано, видоизмените файл `init.js` таким образом, чтобы строка запроса, поступившая от ссылки, на которой был выполнен щелчок, сохранялась в переменной `data`. Для этого введите в файл следующий дополнительный код, выделенный полужирным шрифтом.

```
// Удостовериться в готовности документа, прежде чем выполнять сценарии
jQuery(function($){

// Захватывать события в модальном окне
$("li>a").live("click", function(event){

      // Предотвратить загрузку файла view.php по щелчку на ссылке
      event.preventDefault();

      // Добавить в ссылку класс "active"
      $(this).addClass("active");

      // Получить строку запроса из атрибута "href" ссылки
      var data = $(this)
```

```
                    .attr("href")
                    .replace(/.+?\?(.*)$/, "$1");
      // Вывести строку запроса
      console.log( data );

    });

});
```

Сохраните изменения, а затем загрузите страницу `http://localhost/` и щелкните на любой ссылке. Вы должны увидеть в консоли примерно такой результат.

```
event_id=1
```

## Создание модального окна

Далее нам предстоит сгенерировать HTML-разметку, которая и будет создавать модальное окно. Эта разметка довольно проста и в основном будет представлена одним элементом `div`, служащим оболочкой для другого содержимого. Например, разметка модального окна для события *New Year's Day* выглядит так.

```
<div class="modal-window">
    <h2>New Year's Day</h2>
    <p class="dates">January 01, 2010, 12:00am—11:59pm</p>
    <p>Happy New Year!</p>
</div>
```

Такие же модальные окна будут использоваться и в других случаях (например, для отображения формы, предназначенной для редактирования событий), поэтому мы абстрагируем процедуру фактического создания модального окна в виде отдельной функции, которую можно будет многократно использовать. Поскольку мы собираемся повторно использовать не только эту функцию, соберем все аналогичные вспомогательные функции, используемые в нашем сценарии, в один *объектный литерал*, представляющий собой список пар *имя-значение*, разделенных точкой с запятой (более подробно об этом говорится далее).

### Создание вспомогательной функции для проверки существования модального окна

Поместите в начале файла `init.js` объявление нового литерального объекта `fx`, предназначенного для хранения вспомогательных функций.

```
// Удостовериться в готовности документа, прежде чем выполнять сценарии
jQuery(function($){

// Функции для манипулирования модальным окном
var fx = {};

// Захватывать события в модальном окне
$("li>a").live("click", function(event){

      // Предотвратить загрузку файла view.php по щелчку на ссылке
      event.preventDefault();

      // Добавить в ссылку класс "active"
```

```
    $(this).addClass("active");

    // Получить строку запроса из атрибута "href" ссылки
    var data = $(this)
                    .attr("href")
                    .replace(/.+?\?(.*)$/, "$1");
    // Вывести строку запроса
    console.log( data );

  });

});
```

Первая из функций, которые мы сохраним в объекте `fx`, будет называться `initModal`. Она предназначена для проверки того, что модальное окно уже существует. Если это действительно так, то функция выбирает это окно, в противном случае она создает новое окно и присоединяет его к дескриптору `body`.

Чтобы проверить существование элемента, можно воспользоваться значением его свойства `length` после вызова функции jQuery с селектором для данного элемента. Если значение свойства `length` равно `0`, то это означает, что в настоящее время указанный элемент отсутствует в объектной модели документа (DOM).

Чтобы выполнить описанную проверку и возвратить модальное окно, добавьте в объект `fx` в файле `init.js` следующий код, выделенный полужирным шрифтом.

```
// Функции для манипулирования модальным окном
var fx = {

        // Возвращает модальное окно, если оно существует;
        // в противном случае создает новое модальное окно
        "initModal" : function() {
                // Если подходящие элементы отсутствуют, свойство
                // length возвратит значение 0
                if ( $(".modal-window").length==0 )
                {
                    // Создать элемент div, добавить класс и
                    // присоединить его к дескриптору body
                    return $("<div>")
                            .addClass("modal-window")
                            .appendTo("body");
                }
                else
                {
                    // Возвратить модальное окно, если оно уже существует в DOM
                    return $(".modal-window");
            }
        }

};
```

### Вызов вспомогательной функции из обработчика событий

Далее мы модифицируем обработчик события `click` для загрузки результата вызова `fx.initModal` в переменную, которая будет использоваться в сценарии, добавив для этого в файл `init.js` следующий код, выделенный полужирным шрифтом.

```
// Захватывать события в модальном окне
$("li>a").live("click", function(event){

      // Предотвратить загрузку файла view.php по щелчку на ссылке
      event.preventDefault();

      // Добавить в ссылку класс "active"
      $(this).addClass("active");

      // Получить строку запроса из атрибута "href" ссылки
      var data = $(this)
                     .attr("href")
                     .replace(/.+?\?(.*)$/, "$1"),

      // Проверить существование модального окна и выбрать
      // его или создать новое окно
      modal = fx.initModal();
    });
```

**Примечание.** В этом примере точка с запятой (`;`), стоящая в конце строки с переменной `data`, заменена запятой (`,`).

Сохраните файл, перезагрузите страницу `http://localhost/` и щелкните на одном из названий событий для вызова модального окна (рис. 7.1).

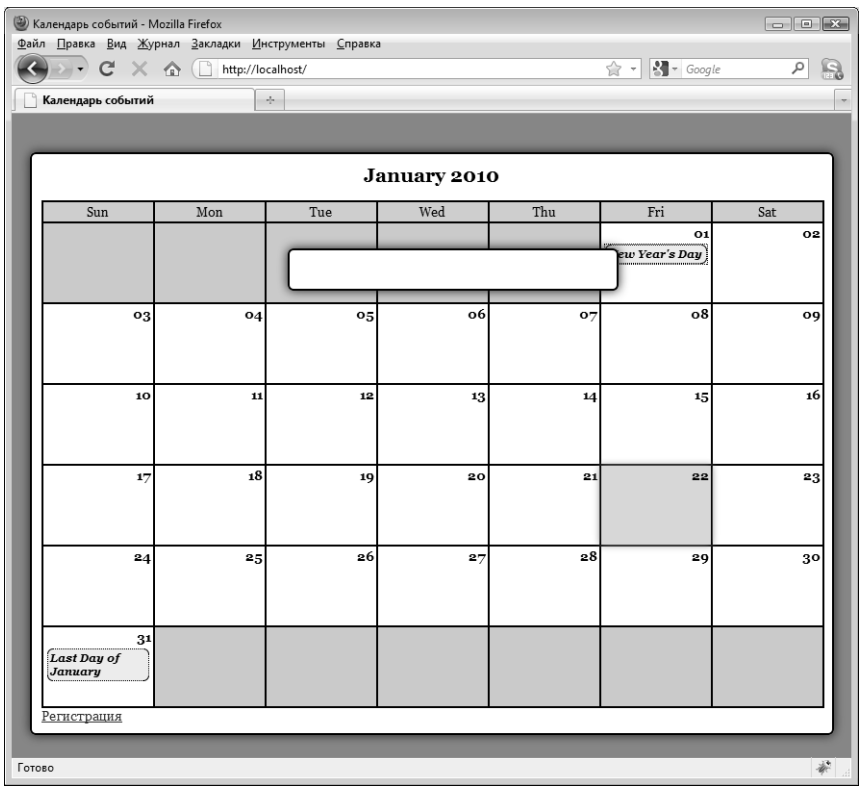



**Рис. 7.1.** Появление модального окна после щелчка на названии события

### Использование объектного литерала для хранения вспомогательных функций

В процессе написания сценариев часто приходится использовать вспомогательные функции. Чем сложнее приложение, тем больше вероятность того, что оно будет нуждаться в значительном количестве вспомогательных функций, и тем труднее содержать эти функции в организованном порядке.

Одним из возможных способов организации вспомогательных функций является использование объектных литералов. Это позволяет разместить все функции в одном месте и даже сгруппировать их в соответствии с назначением.

#### Объектные литералы

В своей простейшей форме объектный литерал — это переменная JavaScript с двумя фигурными скобками после нее, обозначающими пустой объектный литерал.

```
var obj = {};
```

В объектный литерал можно добавить любое количество значений, используя пары *имя-значение*, разделенные запятой.

```
var obj = {
    "name" : "Jason Lengstorf",
    "age" : "25"
};
```

Чтобы получить доступ к какому-либо значению, достаточно добавить к имени переменной точку (.) и имя нужного свойства.

```
alert(obj.name); // выводит сообщение "Jason Lengstorf"
```

Что делает литералы особенно ценными, так это то, что в них можно хранить также функции.

```
var obj = {
  "func" : function() { alert("Объектные литералы -- это сила!"); }
};
```

Для вызова функций, сохраненных в литерале, используется тот же синтаксис, что и для доступа к значениям, с тем лишь отличием, что в конце необходимо добавлять пару круглых скобок. В противном случае JavaScript будет предполагать, что вы пытаетесь сохранить функцию в другой переменной, и просто возвратит ее.

```
obj.func(); // выводит сообщение " Объектные литералы -- это сила!"
```

Кроме того, функции в объектных литералах могут принимать параметры.

```
var obj = {
  "func" : function(text){ alert(text); }
};
obj.func("Это параметр!"); // выводит сообщение " Это параметр!"
```

#### Объектные литералы и процедурное программирование

Организованное хранение функций в литералах обеспечивает ясность кода, а если разработчик прилагает усилия к тому, чтобы функции были достаточно абстрактными, то это еще и сокращает время, которое приходится тратить на сопровождение кода, поскольку все в нем хранится на своих местах и легко находится.

Несмотря на это, объектные литералы не всегда являются самым оптимальным решением. В тех случаях, когда приходится иметь дело со множеством объектов, лучше всего использовать полностью объектно-ориентированный подход. Если сценарии JavaScript вообще используются лишь в малой степени, то привлечение объектных литералов будет неоправданным.

В конечном счете, только вы, как разработчик, принимаете решение относительно того, какой подход будет наилучшим для конкретного проекта. Это дело вкуса и удобства; вы должны самостоятельно определить, в каком случае процесс разработки будет самым легким.

# Извлечение и отображение информации о событиях с помощью AJAX

Теперь, когда модальное окно загружено, можно загрузить в него информацию о событии и отобразить ее. Для этого воспользуемся методом `$.ajax()`.

Используя метод `$.ajax()`, отправим данные обрабатывающему файлу (который будет создан в следующем разделе), используя метод POST, а затем поместим ответ в модальное окно.

## *Создание файла для обработки запросов AJAX*

Прежде чем мы сведем воедино все необходимое для организации вызова `$.ajax()`, не помешает заранее знать, куда и как именно будут отправляться данные. Создайте в папке `inc` новый файл с именем `ajax.inc.php` (`public/assets/inc/ajax.inc.php`). В значительной степени он будет играть ту же роль, что и файл `process.inc.php`, за исключением того, что в нем будут фигурировать исключительно вызовы AJAX. Поскольку значение, возвращенное PHP-функцией, может быть прочитано JavaScript только в том случае, если оно действительно выведено (с помощью оператора `echo` или аналогичного ему), то файл `process.inc.php` не в состоянии справиться с указанной задачей.

По сути, файл `ajax.inc.php` будет использовать поисковый массив для того, чтобы определить, какие объекты и методы следует использовать, а затем выводить возвращенные ему значения с помощью оператора `echo` для использования в AJAX.

Начнем с запуска сеанса, загрузки необходимой конфигурационной информации, определения констант и сборки всего, что необходимо, с помощью функции автозагрузки. С этой целью добавьте в файл `ajax.inc.php` следующий код.

```
<?php

/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
```

```
        include_once $filename;
    }
}

?>
```

Определите поисковый массив с информацией для загрузки данных о событиях, а затем объедините весь код, предназначенный для создания экземпляра объекта, вызова метода и вывода возвращенного значения, внеся для этого изменения, выделенные ниже полужирным шрифтом.

```
<?php

/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_view' => array(
                'object' => 'Calendar',
                'method' => 'displayEvent'
            )
    );

/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);

    /*
     * Проверить наличие идентификатора ID и выполнить необходимую коррекцию
     */
    if ( isset($_POST['event_id']) )
    {
```

```
        $id = (int) $_POST['event_id'];
    }
    else { $id = NULL; }

    echo $obj->$use_array['method']($id);
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}

?>
```

Единственным существенным отличием приведенного выше кода от файла `process.inc.php` является отсутствие ключа `header` в поисковом массиве и использование оператора `echo` для вывода значений, возвращаемых вызываемыми методами.

### Загрузка информации о событии с помощью AJAX

Вновь обратившись к файлу `init.js`, добавим в него вызов `$.ajax()`. Далее в приложении будут другие вызовы функции `$.ajax()`, поэтому имеет смысл сохранить расположение обрабатывающего файла в переменной, чтобы имя файла или его местоположение можно было легко изменить, если в этом возникнет необходимость. Добавьте эту переменную в файл `init.js`, введя в него следующий код, выделенный полужирным шрифтом.

```
// Удостовериться в готовности документа, прежде чем выполнять
// сценарии
jQuery(function($){

// Файл, которому следует отправить запрос AJAX
var processFile = "assets/inc/ajax.inc.php",

// Функции для манипулирования модальным окном
    fx = {

        // Возвращает модальное окно, если оно существует;
        // в противном случае создает новое модальное окно
        "initModal" : function() {
                // Если подходящие элементы отсутствуют, свойство
                // length возвратит значение 0
                if ( $(".modal-window").length==0 )
                {
                    // Создать элемент div, добавить класс и
                    // присоединить его к дескриптору body
                    return $("<div>")
                            .addClass("modal-window")
                            .appendTo("body");
```

```
                }
                else
                {
                    // Возвратить модальное окно, если оно уже существует в DOM
                    return $(".modal-window");
            }
        }
    };

// Захватывать события в модальном окне
$("li>a").live("click", function(event){

      // Предотвратить загрузку файла view.php по щелчку на ссылке
      event.preventDefault();

      // Добавить в ссылку класс "active"
      $(this).addClass("active");

      // Получить строку запроса из атрибута "href" ссылки
      var data = $(this)
                     .attr("href")
                     .replace(/.+?\?(.*)$/, "$1"),

      // Проверить существование модального окна и выбрать
      // его или создать новое окно
      modal = fx.initModal();

    });

});
```

После этого настройте вызов метода `$.ajax()` в обработчике события. Он будет использовать метод POST, указывать на переменную `processFile` и отправлять соответствующие данные. Поскольку в строку запроса, извлеченную из ссылки, не входит поле `action`, вставьте его здесь вручную. Наконец, используйте метод `.append()` для вставки возвращенной разметки в модальное окно в случае успешного выполнения вызова или, в противном случае, для отображения сообщения об ошибке.

Вставьте в файл `init.js` следующий код, выделенный полужирным шрифтом.

```
// Захватывать события в модальном окне
$("li>a").live("click", function(event){

        // Предотвратить загрузку файла view.php по щелчку на ссылке
        event.preventDefault();

        // Добавить в ссылку класс "active"
        $(this).addClass("active");

        // Получить строку запроса из атрибута "href" ссылки
        var data = $(this)
                       .attr("href")
                       .replace(/.+?\?(.*)$/, "$1"),

        // Проверить существование модального окна и выбрать
        // его или создать новое окно
```

```
    modal = fx.initModal();

    // Загрузить информацию о событии из БД
    $.ajax({
            type: "POST",
            url: processFile,
            data: "action=event_view&" + data,
            success: function(data){
                    // Пока только вывести информацию о событии
                    modal.append(data);
                },
            error: function(msg) {
                    modal.append(msg);
                }
        });

});
```

Сохраните изменения, перезагрузите страницу `http://localhost/` и щелкните на названии события, в результате чего информация о событии загрузится в модальное окно (рис. 7.2).

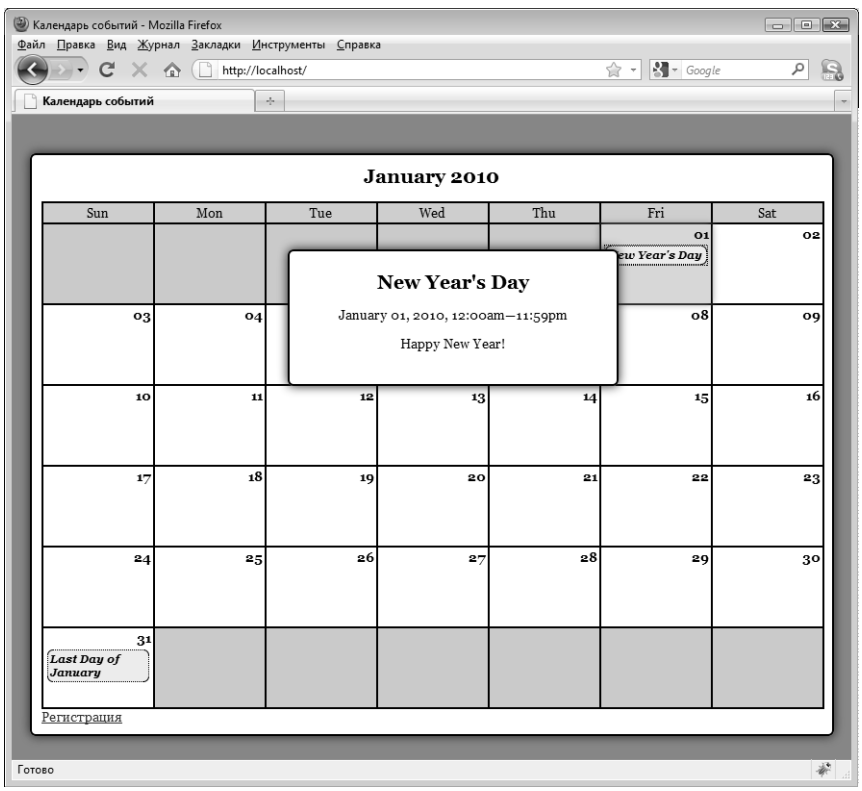



**Рис. 7.2.** Информация о событии загружена в модальное окно

## Добавление кнопки закрытия окна

В том состоянии, в котором приложение находится на данном этапе, единственным способом избавиться от модального окна, открывающегося при щелчке на названии события, является перезагрузка страницы. Разумеется, это совсем не то, что надо, поэтому добавим кнопку закрытия окна.

Для этого создадим новую ссылку и свяжем с ней обработчик событий, удаляющий модальное окно из DOM. Чтобы придать кнопке Закрыть традиционный внешний вид, отобразим на ней символ × (тогда как стиль, определенный в файле `ajax.css`, добавит перед ней слово "Закрыть"). Кроме того, добавим в кнопку атрибут `href`, в результате чего при наведении указателя на кнопку ее внешний будет изменяться, указывая на то, что она способна реагировать на щелчки.

Чтобы добавить кнопку закрытия модального окна, внесите в файл `init.js` изменения, выделенные ниже полужирным шрифтом.

```
// Захватывать события в модальном окне
$("li>a").live("click", function(event){

        // Предотвратить загрузку файла view.php по щелчку на ссылке
        event.preventDefault();

        // Добавить в ссылку класс "active"
        $(this).addClass("active");

        // Получить строку запроса из атрибута "href" ссылки
        var data = $(this)
                        .attr("href")
                        .replace(/.+?\?(.*)$/, "$1"),

        // Проверить существование модального окна и выбрать
        // его или создать новое окно
        modal = fx.initModal();

        // Создать кнопку для закрытия окна
        $("<a>")
            .attr("href", "#")
            .addClass("modal-close-btn")
            .html("&times;")
            .click(function(event){
                        // Предотвратить выполнение действия по умолчанию
                        event.preventDefault();

                        // Удалить модальное окно
                        $(".modal-window")
                            .remove();
                    })
            .appendTo(modal);

        // Загрузить информацию о событии из БД
        $.ajax({
                type: "POST",
                url: processFile,
                data: "action=event_view&" + data,
                success: function(data){
```

```
                // Пока только вывести информацию о событии
                modal.append(data);
            },
        error: function(msg) {
                modal.append(msg);
            }
    });

});
```

Сохраните код, загрузите страницу `http://localhost/` и щелкните на названии события, чтобы проконтролировать появление кнопки Закрыть (рис. 7.3). Щелчок на этой кнопке приведет к закрытию модального окна.

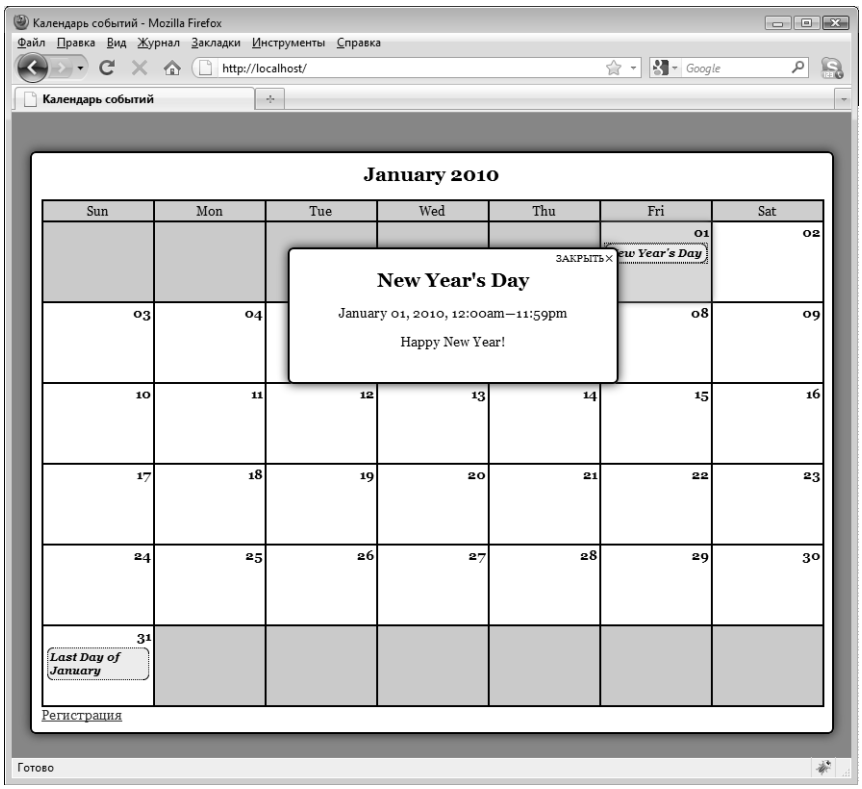



**Рис. 7.3.** Теперь в модальном окне отображается кнопка Закрыть

## Добавление эффектов в процессы создания и уничтожения модального окна

Чтобы придать нашему модальному окну определенный лоск, добавим эффекты, благодаря которым оно будет плавно появляться при открытии и так же плавно исчезать при закрытии. Кроме того, чтобы внимание пользователя фокусировалось на модальном окне, когда оно активно, добавим оверлей, затемняющий всю остальную часть страницы.

### Эффект плавного исчезновения модального окна

Прежде всего, мы добавим эффект, обеспечивающий плавное исчезновение модального окна (постепенное уменьшение его видимости до полного закрытия). Эта функция будет запускаться несколькими методами, причем некоторые из них будут запускать также и события. Для этого введем в код условный оператор, который проверяет, запущено ли событие, и, если это действительно так, предотвращает выполнение действия, предусмотренного по умолчанию.

Затем мы удалим класс `active` из всех ссылок, поскольку ни одна из них не может быть использована, пока модальное окно остается невидимым.

Наконец, выберем модальное окно и обеспечим его плавное исчезновение с помощью функции `.fadeOut()`. В ее функции обратного вызова модальное окно полностью удаляется из DOM.

Чтобы добавить эту функцию, вставьте в объект `fx` следующий код, выделенный полужирным шрифтом.

```
// Функции для манипулирования модальным окном
    fx = {

        // Возвращает модальное окно, если оно существует;
        // в противном случае создает новое модальное окно
        "initModal" : function() {
                // Если подходящие элементы отсутствуют, свойство
                // length возвратит значение 0
                if ( $(".modal-window").length==0 )
                {
                    // Создать элемент div, добавить класс и
                    // присоединить его к дескриптору body
                    return $("<div>")
                            .addClass("modal-window")
                            .appendTo("body");
                }
                else
                {
                    // Возвратить модальное окно, если оно уже существует в DOM
                    return $(".modal-window");
                }
            },
        // Обеспечивает плавное исчезновение окна и его удаление из DOM
        "boxout" : function(event) {
                // Если событие было запущено элементом, который
                // вызвал эту функцию, предотвратить выполнение
                // действия, заданного по умолчанию
                if ( event!=undefined )
                {
                    event.preventDefault();
                }
                // Удалить класс "active" из всех ссылок
                $("a").removeClass("active");
                // Обеспечить плавное исчезновение модального окна,
                // а затем полностью удалить его из DOM
                $(".modal-window")
                    .fadeOut("slow", function() {
                            $(this).remove();
```

```
                }
            );
        }

};
```

Чтобы включить новую функцию в сценарий, видоизменим обработчик события щелчка для кнопки Закрыть, добавив код, выделенный ниже полужирным шрифтом.

```
// Создать кнопку для закрытия окна
$("<a>")
    .attr("href", "#")
    .addClass("modal-close-btn")
    .html("&times;")
    .click(function(event){
            // Удалить модальное окно
            fx.boxout(event);
        })
    .appendTo(modal);
```

Сохраните файл `init.js` и перезагрузите страницу `http://localhost/` в браузере. Щелкните сначала на названии события для создания нового модального окна, а затем — на кнопке Закрыть, и вы увидите, как модальное окно плавно исчезнет (рис. 7.4).

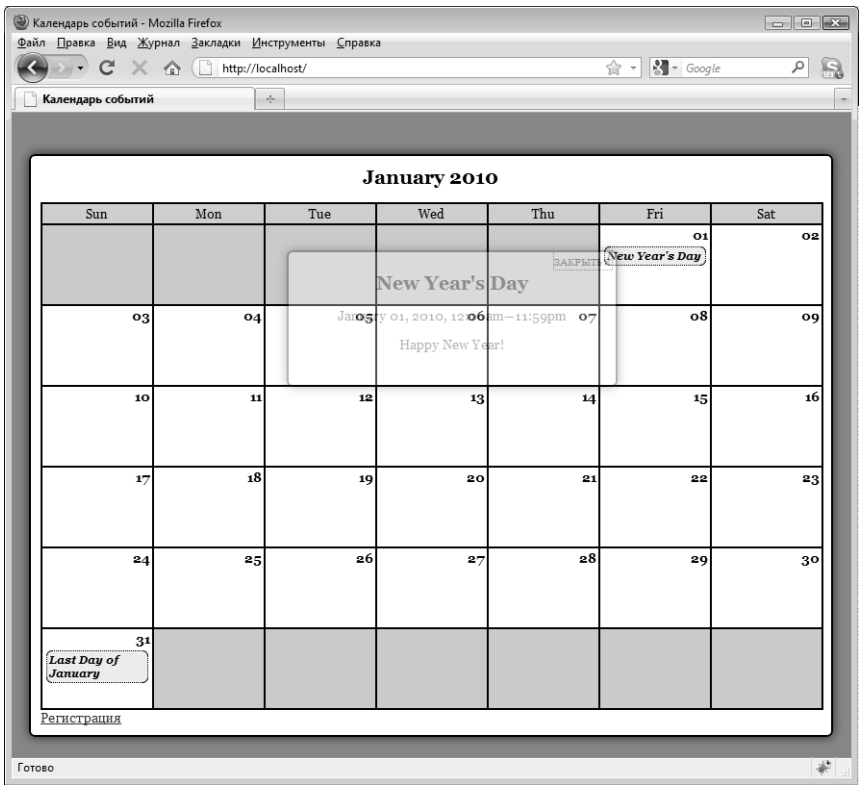



**Рис. 7.4.** Модальное окно в процессе плавного исчезновения после щелчка на кнопке Закрыть

### Добавление эффектов оверлея и плавного появления модального окна

Чтобы включить в приложение эффекты оверлея и плавного появления модального окна, потребуется добавить в объектный литерал `fx` еще одну функцию, которую назовем `boxin`. Она будет вызываться из функции обратного вызова `$.ajax()` в обработчике события щелчка на названии и принимать два параметра: данные, возвращенные файлом `ajax.inc.php` (`data`), и объект модального окна (`modal`).

Прежде всего функция создает новый контейнер `div` с классом `modal-overlay`, а затем скрывает элемент `div` и присоединяет его к телу документа. Для придания оверлею дополнительной функциональности к нему будет присоединен обработчик события щелчка, удаляющий модальное окно путем вызова функции `fx.boxout()` при щелчке на оверлее.

После этого функция создает модальное окно и присоединяет к нему информацию, хранящуюся в переменной `data`. Наконец, она обеспечивает плавное появление на экране обоих элементов с помощью функции `.fadeIn()`.

Чтобы добавить эту функцию в объектный литерал `fx`, введите следующий код, выделенный полужирным шрифтом.

```
// Функции для манипулирования модальным окном
    fx = {

        // Возвращает модальное окно, если оно существует;
        // в противном случае создает новое модальное окно
        "initModal" : function() {
                // Если подходящие элементы отсутствуют, свойство
                // length возвратит значение 0
                if ( $(".modal-window").length==0 )
                {
                    // Создать элемент div, добавить класс и
                    // присоединить его к дескриптору body
                    return $("<div>")
                            .addClass("modal-window")
                            .appendTo("body");
                }
                else
                {
                    // Возвратить модальное окно, если оно уже существует в DOM
                    return $(".modal-window");
                }
            },

        // Добавляет окно в разметку и обеспечивает его плавное появление
        "boxin" : function(data, modal) {
                // Создать оверлей для сайта, добавить класс и обработчик
                // события щелчка и присоединить их к телу документа
                $("<div>")
                    .hide()
                    .addClass("modal-overlay")
                    .click(function(event){
                            // Удалить событие
                            fx.boxout(event);
                        })
                    .appendTo("body");
```

```
            // Загрузить данные в модальное окно
            // и присоединить его к телу документа
            modal
                .hide()
                .append(data)
                .appendTo("body");

            // Обеспечить плавное появление модального окна и оверлея
            $(".modal-window,.modal-overlay")
                .fadeIn("slow");

        },

    // Обеспечивает плавное исчезновение окна и его удаление из DOM
    "boxout" : function(event) {
            // Если событие было запущено элементом, который
            // вызвал эту функцию, предотвратить выполнение
            // действия, заданного по умолчанию
            if ( event!=undefined )
            {
               event.preventDefault();
            }

            // Удалить класс "active" из всех ссылок
            $("a").removeClass("active");

            // Обеспечить плавное исчезновение модального окна,
            // а затем полностью удалить его из DOM
            $(".modal-window")
            $(".modal-window,.modal-overlay")
                .fadeOut("slow", function() {
                        $(this).remove();
                    }
                );
        }

};
```

После этого необходимо модифицировать функцию обратного вызова, которая запускается в случае успешного выполнения функции `$.ajax()` при щелчке на названии события, чтобы вызвать функцию `fxboxin`. Для этого добавьте в код строку, выделенную в листинге полужирным шрифтом.

```
// Захватывать события в модальном окне
$("li>a").live("click", function(event){
        // Предотвратить загрузку файла view.php по щелчку на ссылке
        event.preventDefault();

        // Добавить в ссылку класс "active"
        $(this).addClass("active");

        // Получить строку запроса из атрибута href ссылки
        var data = $(this)
                        .attr("href")
                        .replace(/.+?\?(.*)$/, "$1"),
```

```
        // Проверить существование модального окна и выбрать
        // его или создать новое окно
            modal = fx.initModal();

        // Создать кнопку для закрытия окна
        $("<a>")
            .attr("href", "#")
            .addClass("modal-close-btn")
            .html("&times;")
            .click(function(event){
                        // Удалить модальное окно
                        fx.boxout(event);
                    })
            .appendTo(modal);

        // Загрузить информацию о событии из БД
        $.ajax({
                type: "POST",
                url: processFile,
                data: "action=event_view&" + data,
                success: function(data){
                        fx.boxin(data, modal);
                    },
                error: function(msg) {
                        modal.append(msg);
                    }
            });

    });
```

Сохраните этот код, перезагрузите страницу `http://localhost/` и щелкните на названии события, чтобы увидеть в действии эффект плавного появления модального окна и модального оверлея (рис. 7.5).

Возможно, вы заметили, что в момент открытия модального окна наблюдается легкое дрожание изображения. Это происходит потому, что функция `fx.initModal()` присоединяет модальное окно к телу документа без его предварительного сокрытия. Чтобы избавиться от этого недостатка, добавьте вызов функции `.hide()` в функцию `fx.initModal()` с помощью следующего кода, выделенного полужирным шрифтом.

```
// Функции для манипулирования модальным окном
    fx = {

        // Возвращает модальное окно, если оно существует;
        // в противном случае создает новое модальное окно
        "initModal" : function() {
                // Если подходящие элементы отсутствуют, свойство
                // length возвратит значение 0
                if ( $(".modal-window").length==0 )
                {
                    // Создать элемент div, добавить класс и
                    // присоединить его к дескриптору body
                    return $("<div>")
                            .hide()
                            .addClass("modal-window")
                            .appendTo("body");
```

```
            }
            else
            {
                // Возвратить модальное окно, если оно уже существует в DOM
                return $(".modal-window");
            }
        },

    // Добавляет окно в разметку и обеспечивает его плавное появление
    "boxin" : function(data, modal) {
            // Для краткости код опущен
        },

    // Обеспечивает плавное исчезновение окна и его удаление из DOM
    "boxout" : function(event) {
            // Для краткости код опущен
        }

};
```

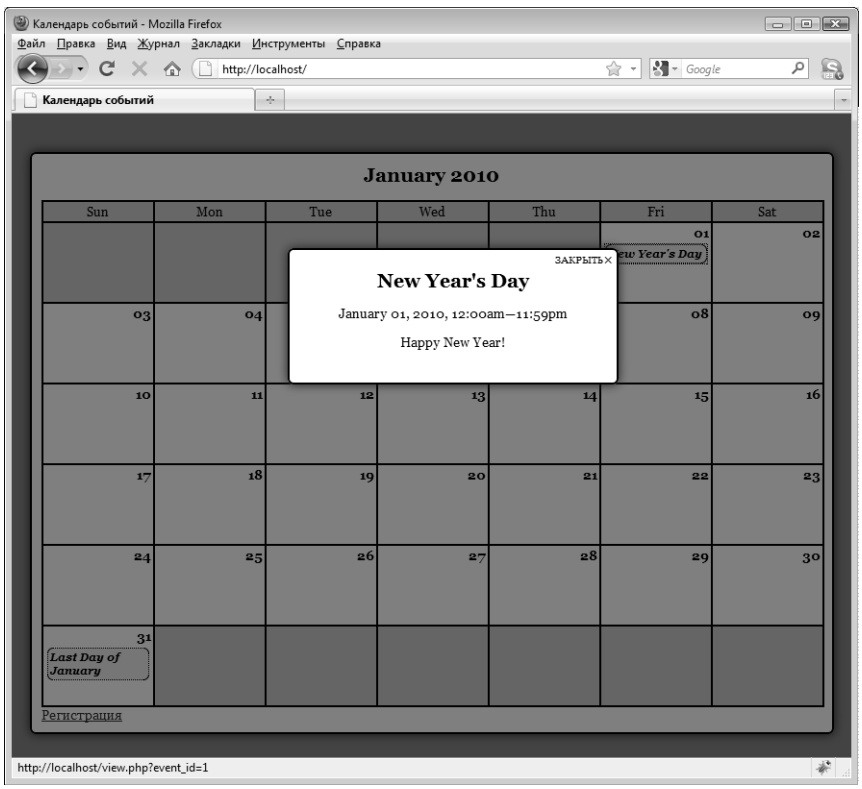



**Рис. 7.5.** Использование модального окна с оверлеем фокусирует внимание на отображаемой информации

Наконец, заметим, что щелчок на кнопке Закрыть не приводит к удалению оверлея. Для создания эффекта плавного исчезновения и удаления оверлея достаточно изменить селектор в функции `fx.boxout()`.

```
// Функции для манипулирования модальным окном
    fx = {

        // Возвращает модальное окно, если оно существует;
        // в противном случае создает новое модальное окно
        "initModal" : function() {
                // Для краткости код опущен
            },

        // Добавляет окно в разметку и обеспечивает его плавное
        // появление
        "boxin" : function(data, modal) {
                // Для краткости код опущен
            },

        // Обеспечивает плавное исчезновение окна и его удаление из DOM
        "boxout" : function(event) {
                // Если событие было запущено элементом, который
                // вызвал эту функцию, предотвратить выполнение
                // действия, заданного по умолчанию
                if ( event!=undefined )
                {
                   event.preventDefault();
                }

                // Удалить класс active из всех ссылок
                $("a").removeClass("active");

                // Обеспечить плавное исчезновение модального окна
                // и оверлея, а затем полностью удалить их из DOM
                $(".modal-window,.modal-overlay")
                    .fadeOut("slow", function() {
                            $(this).remove();
                        }
                    );
            }

    };
```

Внеся это изменение, перезагрузите страницу `http://localhost/` и щелкните на названии события. Модальное окно и оверлей плавно появятся на экране, а после щелчка на кнопке Закрыть или на оверлее — плавно исчезнут.

# Резюме

В этой главе вы научились динамически загружать информацию о событиях средствами jQuery, придерживаясь принципов прогрессивного улучшения. Вы также познакомились с методами обработки событий, элементарными эффектами и даже с регулярными выражениями.

В следующей главе процесс добавления функциональности AJAX в приложение будет продолжен, и возможности AJAX будут распространены также на элементы управления, обеспечивающие редактирование событий.

# Глава 8

# Редактирование данных календаря средствами AJAX и jQuery

Теперь, когда приложение способно отображать информацию о событиях без обновления страницы, вы имеете возможность самостоятельно убедиться в том, какие преимущества дает использование AJAX в веб-приложениях. Так уж исторически сложилось, что одним из наибольших недостатков веб-приложений было то, что после любого, даже самого незначительного действия приходилось дожидаться, пока страница не обновится полностью, прежде чем сохранятся результаты. Веб-приложения были удобны в тех случаях, когда пользователь нуждался в информации, хранящейся на общем компьютере, но медленная работа подобных приложений служила достаточным основанием для того, чтобы пользователи при малейшей возможности отдавали предпочтение их настольным аналогам.

Вместе с тем, теперь, когда эффективность AJAX признана всем сообществом пользователей, появилась возможность быстро вносить изменения, не дожидаясь перезагрузки страницы. Это приближает поведение веб-приложений к поведению настольных, тем самым повышая их привлекательность в глазах пользователей.

В этой главе вы узнаете о том, как добавить в приложение сценарии, благодаря которым административные элементы управления будут работать бесперебойно, не требуя полного обновления страницы. Единственным случаем, требующим обновления страницы, будет вход в приложение, поскольку с этим связаны изменения на уровне сеанса.

**Примечание.** Прежде чем приступить к выполнению упражнений, приведенных в данной главе, пройдите процедуру регистрации в приложении. По умолчанию в качестве имени пользователя используется `testuser`, а в качестве пароля — `admin`.

## Открытие формы для создания событий

Начнем с того, что внесем в сценарий изменения, позволяющие администраторам добавлять новые события без обновления страницы. Откройте файл `init.js` и выберите с помощью селектора кнопку для добавления событий, используя в качестве критерия отбора ее класс (`admin`)[1]. Добавьте обработчик события `click`,

[1] При этом селектор выберет и кнопку Выход, которая также имеет класс `admin`, в результате чего она перестанет выполнять свои прямые функции. Чтобы восстановить функциональность этой кнопки, отмените назначение класса `admin` в ее определении (см. главу 6) или добавьте в самом начале кода обработчика событий `function(event)` соответствующую проверку. — *Примеч. ред.*

предотвращающий выполнение действия, предусмотренного по умолчанию, и (только в данном примере) выводящий сообщение, позволяющее убедиться в том, что обработчик действительно был запущен.

```
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",
    fx = {
        "initModal" : function() {...},
        "boxin" : function(data, modal) {...},
        "boxout" : function(event) {...}
    };

     $("li>a").live("click", function(event){...});

// Отображает форму для редактирования в виде модального окна
$(".admin").live("click", function(event){

        // Предотвратить отправку формы
        event.preventDefault();

       // Вывести сообщение для проверки запуска обработчика
        console.log( "Выполнен щелчок на кнопке добавления нового события!" );
    });
});
```

**Примечание.** В данной главе в примерах кода все функции, в которые не вносятся изменения, для краткости отображаются в свернутом виде, а комментарии опускаются. Полная версия кода находится на веб-странице данной книги на сайте Apress по такому адресу `http://apress.com/book/view/1430228474`.

Сохраните изменения и обновите страницу `http://localhost/`. После щелчка на кнопке Добавить новое событие в окне консоли должно отобразиться следующее сообщение.

```
Выполнен щелчок на кнопке добавления нового события!
```

## Добавление вызова AJAX для загрузки формы

Далее создайте переменную для хранения значения атрибута `action` формы, которое будет отправлено обрабатывающему файлу. Поскольку мы загружаем форму, предназначенную для редактирования и создания событий, присвойте этой переменной значение `event_edit`.

Теперь можно вызвать функцию `$.ajax()`. Она будет аналогична сценарию для загрузки информации о событии в модальное окно; в действительности оба этих случая будут различаться лишь набором отправляемых данных и способом обработки возвращаемого значения.

В случае успешной загрузки необходимо скрыть форму и сохранить ссылку на нее в переменной `form`. Далее необходимо убедиться в существовании модального окна с помощью функции `fx.initModal()` и обеспечить его плавное появление вызовом функции `fx.boxin()`, передав ей в качестве первого параметра значение `null`. Наконец, необходимо присоединить форму к модальному окну, обеспечить его плавное появление и назначить ему класс `edit-form`, чтобы упростить в дальнейшем его выбор.

Для выполнения этих действий добавьте в файл `init.js` следующий код, выделенный полужирным шрифтом.

```
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",
    fx = {
        "initModal" : function() {...},
        "boxin" : function(data, modal) {...},
        "boxout" : function(event) {...}
    };

    $("li>a").live("click", function(event){...});

    // Отображает форму для редактирования в виде модального окна
    $(".admin").live("click", function(event){

       // Предотвратить отправку формы
       event.preventDefault();

       // Загрузить атрибут action для обрабатывающего файла
       var action = "edit_event";

       // Загрузить форму для редактирования событий и отобразить ее
       $.ajax({
             type: "POST",
             url: processFile,
             data: "action="+action,
             success: function(data){
                    // Скрыть форму
                    var form = $(data).hide(),

                    // Убедиться в существовании модального окна
                    modal = fx.initModal();

                    // Вызвать функцию boxin для создания модального
                    // оверлея и обеспечить его плавное появление
                    fx.boxin(null, modal);

                    // Загрузить форму в окно, обеспечить плавное
                    // появление содержимого и добавить класс в форму
                    form
                        .appendTo(modal)
                        .addClass("edit-form")
                        .fadeIn("slow");
             },
             error: function(msg){
                 alert(msg);
             }
       });

    });

});
```

## Модификация обрабатывающего файла AJAX для загрузки формы

Прежде чем предыдущий вызов AJAX заработает, необходимо изменить поисковый массив в файле `ajax.inc.php`. Добавьте в массив новый элемент, указывающий сценарию на необходимость создания нового объекта `Calendar`, а затем вызовите метод `displayForm()`, внеся в код изменения, выделенные ниже полужирным шрифтом.

```
<?php

/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_view' => array(
                'object' => 'Calendar',
                'method' => 'displayEvent'
            ),
        'edit_event' => array(
                'object' => 'Calendar',
                'method' => 'displayForm'
            )
    );

/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);

    /*
     * Проверить наличие идентификатора ID и выполнить необходимую коррекцию
     */
    if ( isset($_POST['event_id']) )
```

```
    {
        $id = (int) $_POST['event_id'];
    }
    else { $id = NULL; }

    echo $obj->$use_array['method']($id);
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}

?>
```

Сохраните файл, перезагрузите страницу `http://localhost/` и щелкните на кнопке Добавить новое событие. На экране должно плавно отобразиться новое модальное окно с формой для создания события (рис. 8.1).

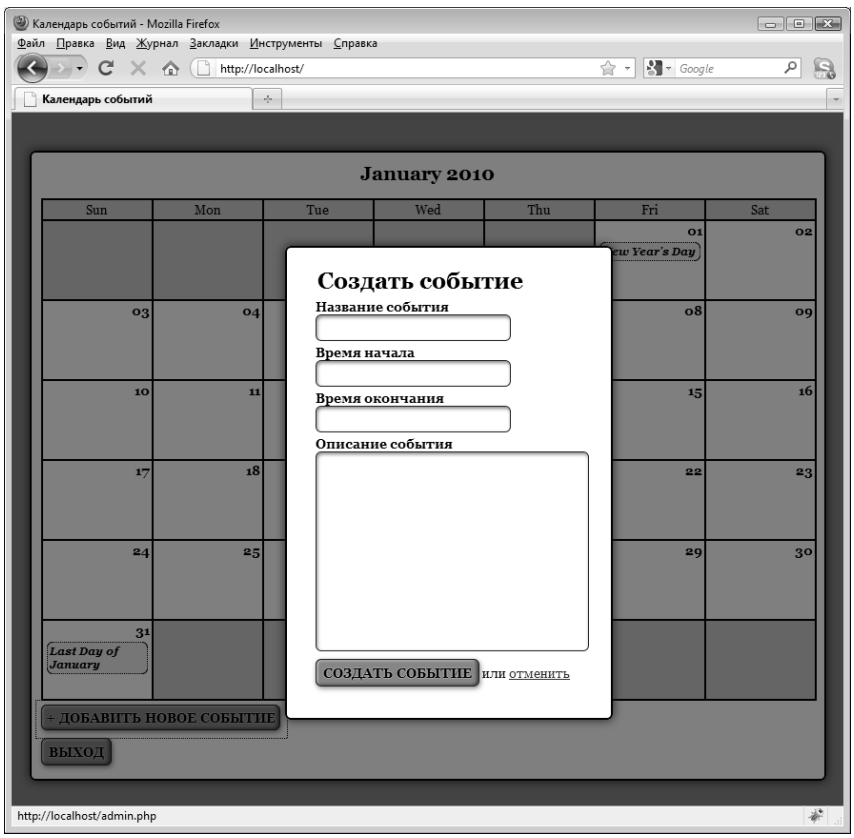



**Рис. 8.1.** Форма для создания события, загруженная в модальное окно

## Закрытие модального окна при щелчке на кнопке Отменить

Возможно, вы обратили внимание на отсутствие кнопки Закрыть при отображении формы в модальном окне. Вместе с тем, в модальном окне имеется кнопка Отменить, щелчок на которой приводит к обновлению страницы.

Вместо того чтобы добавлять в окно еще одну кнопку, достаточно сделать так, чтобы при щелчке на кнопке Отменить вызывался метод `fx.boxout()`, закрывающий окно.

Чтобы добиться этого, используйте функцию `.live()` для привязки обработчика события `click` к любой ссылке, содержащей слово `отменить`, внутри формы с классом `edit-form`.

```
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",
    fx = {
        "initModal" : function() {...},
        "boxin" : function(data, modal) {...},
        "boxout" : function(event) {...}
    };

    $("li>a").live("click", function(event){...});

$(".admin").live("click", function(event){...});

// Наделить кнопку "Отменить" на формах редактирования событий
// функциями кнопки "Закрыть" для плавного закрытия и исчезновения
// модального окна и оверлея
$(".edit-form a:contains(отменить)").live("click", function(event){
        fx.boxout(event);
    });

});
```

Сохраните файл, перезагрузите страницу `http://localhost/` и щелкните на кнопке Добавить новое событие. После того как модальное окно перезагрузится, щелкните на кнопке Отменить формы. Модальное окно и оверлей должны плавно исчезнуть, точно так же, как если бы щелчок был выполнен на кнопке Закрыть.

# Сохранение новых событий в базе данных

Далее, чтобы форма работала так, как требуется, в кнопку отправки формы необходимо добавить обработчик события `click`. Этот обработчик предотвратит предусмотренную по умолчанию отправку формы и вместо этого использует функцию `serialize()` для создания строки запроса на основании входных данных формы. После этого он использует сериализованные данные для отправки формы обрабатывающему файлу `ajax.inc.php` с помощью метода `POST`.

Сначала добавьте новый обработчик события `click` во все элементы ввода типа `submit` в форме, которые имеют класс `edit-form`. Использование функции `.live()` гарантирует, что обработчик будет реагировать и на динамически созданные элементы ввода. Чтобы предотвратить выполнение действия, предусмотренного по умолчанию, можно воспользоваться функцией `event.PreventDefault()`.

Для реализации описанных выше действий введите в файл `init.js` следующий код, выделенный полужирным шрифтом.

```
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",
    fx = {
        "initModal" : function() {...},
        "boxin" : function(data, modal) {...},
        "boxout" : function(event) {...}
    };

    $("li>a").live("click", function(event){...});

$(".admin").live("click", function(event){...});

// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Вывести сообщение для проверки работы сценария
        console.log( "Запущена отправка формы!" );
    });

$(".edit-form a:contains(отменить)")
    .live("click", function(event){...};

});
```

Сохраните файл и перезагрузите календарь в браузере. Щелкните сначала на кнопке Добавить новое событие для открытия модального окна, а затем — на кнопке Создать новое событие для отправки формы. В консоли отобразится следующее сообщение.

```
Запущена отправка формы!
```

## Сериализация данных формы

Прежде чем отправить информацию о событии файлу обработки, ее необходимо преобразовать в строку запроса. К счастью, в jQuery для этого предусмотрен встроенный метод `.serialize()`. Он преобразует входные данные формы в строку, содержащую пары *имя-значение*, разделенные символом амперсанда (`&`).

Внесите в файл `init.js` изменения, обеспечивающие сериализацию данных путем выбора формы, являющейся родительской для элемента ввода, на котором был выполнен щелчок, и последующей сериализации данных. Кроме того, в этом примере выведите данные также в консоль.

```
// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();
```

```
        // Сериализовать данные формы для использования
        // с функцией $.ajax()
        var formData = $(this).parents("form").serialize();

        // Вывести сообщение для проверки работы сценария
        console.log( formData );
});
```

Сохраните предыдущий код и откройте в браузере форму для создания события. Введите следующие тестовые данные:

- **название события** — `Test Event`;
- **время начала** — `2010-01-04 08:00:00`;
- **время окончания** — `2010-01-04 10:00:00`;
- **описание события** — `This is a test description`.

Щелкните на кнопке Создать событие для отправки формы. Вывод в консоли должен выглядеть примерно так (значение маркера сеанса у вас может быть другим).

```
event_title=Test+Event&event_start=2010-01-04+08%3A00%3A00
&event_end=2010-01-04+10%3A00%3A00&event_description=This+
is+a+test+description&event_id=&token=21697f4a2326d41995c1
d6291cd5637ca1bbcfbe&action=event_edit
```

## Отправка сериализованных данных формы обрабатывающему файлу

Сериализованные данные можно отправить обрабатывающему файлу с помощью функции `$.ajax()`.

Данные будут отправляться файлу `ajax.inc.php` с использованием метода `POST`, а после их успешной отправки видимость модального окна и оверлея будет плавно уменьшена с помощью функции `fx.boxout()` до их полного исчезновения. Кроме того, в данном примере будет также выведено контрольное сообщение в консоль. Описанные действия реализуются с помощью кода, выделенного ниже полужирным шрифтом.

```
// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Сериализовать данные формы для использования с функцией $.ajax()
        var formData = $(this).parents("form").serialize();

        // Отправить данные обрабатывающему файлу
        $.ajax({
                type: "POST",
                url: processFile,
                data: formData,
                success: function(data) {
                    // Обеспечить плавное исчезновение модального окна
                    fx.boxout();
```

```
                // Вывести сообщение в консоль
                console.log( "Событие сохранено!" );
            },
            error: function(msg) {
                alert(msg);
            }
        });
});
```

На данном этапе сценарий способен сохранять новые события. Однако сначала требуеотся видоизменить файл `ajax.inc.php` для приема этих данных.

## Модификация обрабатывающего файла AJAX для обработки новых отправок

Подготовка файла `ajax.inc.php` к получению данных, отправленных из формы для редактирования событий, сводится к добавлению нового элемента в поисковый массив.

```
/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_view' => array(
                'object' => 'Calendar',
                'method' => 'displayEvent'
            ),
        'edit_event' => array(
                'object' => 'Calendar',
                'method' => 'displayForm'
            ),
        'event_edit' => array(
                'object' => 'Calendar',
                'method' => 'processForm'
            )
    );
```

```
/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);

    /*
     * Проверить наличие идентификатора ID и выполнить
     * необходимую коррекцию
     */
    if ( isset($_POST['event_id']) )
    {
        $id = (int) $_POST['event_id'];
    }
    else { $id = NULL; }

    echo $obj->$use_array['method']($id);
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}

?>
```

Сохраните этот файл и перезагрузите страницу `http://localhost/`. Затем щелкните на кнопке Добавить новое событие для открытия формы в модальном окне и введите информацию о новом событии, используя следующие данные:

- **название события** — `Test Event`;
- **время начала** — `2010-01-04 08:00:00`;
- **время окончания** — `2010-01-04 10:00:00`;
- **описание события** — `This is a test description`.

Щелкните на кнопке Создать событие. Модальное окно плавно исчезнет, а в консоли отобразится следующее сообщение.

```
Событие сохранено!
```

Обратите внимание: для того чтобы новое событие появилось в календаре, необходимо обновить страницу. Это может сбивать пользователей с толку, поэтому в следующем разделе мы видоизменим приложение таким образом, чтобы вновь созданные события добавлялись в календарь сразу же после успешного сохранения данных.

# Добавление событий без обновления страницы

Добавление новых событий в календарь представляет собой довольно сложный процесс. Для этого, после того как событие сохранено, необходимо выполнить ряд действий:

- десериализовать данные формы;
- создать объекты даты как для текущего отображаемого месяца, так и для нового события;
- убедиться в том, что год и месяц соответствуют новому событию;
- получить идентификатор нового события;
- определить, на какой день месяца приходится событие;
- сгенерировать новую ссылку с подходящими данными о событии и вставить ее в соответствующую ячейку календаря.

Описанная функциональность будет помещена в новый член объектного литерала `fx`, который мы назовем `addevent` и который будет принимать данные, возвращаемые файлом `ajax.inc.php` (`data`), а также сериализованные данные формы (`formData`).

Начните с изменения объектного литерала `fx`, добавив в файл `ini.js` следующий код, выделенный полужирным шрифтом.

```
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",
    fx = {
        "initModal" : function() {...},
        "boxin" : function(data, modal) {...},
        "boxout" : function(event) {...},

        // Добавляет новое событие в разметку после сохранения
        "addevent" : function(data, formData){
                // Сюда будет помещен код для добавления события
            }
    };

    $("li>a").live("click", function(event){...});

$(".admin").live("click", function(event){...});

$(".edit-form input[type=submit]")
.live("click", function(event){...});

$(".edit-form a:contains(отменить)")
    .live("click", function(event){...};

});
```

## Десериализация данных формы

Первое, что необходимо сделать при добавлении нового события, — это десериализовать данные формы. Поскольку это действие может выполняться незави-

симо, мы создадим для него в объектном литерале `fx` дополнительную функцию `deserialize`, принимающую строку (`str`).

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Сюда будет помещен код для добавления события
        },
    // Десериализует строку запроса и возвращает
    // объект события
    "deserialize" : function(str){
            // Сюда будет помещен код десериализации
        }

};
```

Как уже отмечалось, сериализованная строка представляет собой последовательность пар *имя-значение*, где члены одной пары соединены знаком равенства (=), а разные пары отделены друг от друга символами амперсанда (&). Например, две такие сериализованные пары могут иметь следующий вид:

```
name1=value1&name2=value2
```

Для десериализации таких пар прежде всего следует разбить строку на подстроки на каждом символе амперсанда с помощью функции JavaScript `.split()`. Эта функция разбивает строку на пары *имя-значение*, одновременно преобразуя их в массив.

```
Array
(
    0 => "name1=value1",
    1 => "name2=value2"
)
```

После этого организуйте цикл по элементам этого массива. Внутри цикла разбейте каждую пару на подстроки на знаке равенства и сохраните массив в переменной `pairs`. Таким образом, каждая пара *имя-значение* преобразуется в массив, в котором первым элементом является *имя*, а вторым — *значение*. Этот массив подчиняется следующему формату.

```
Array
(
    0 => "name1",
    1 => "value1"
)
```

Сохраните эти значения сначала в переменных `key` и `val` соответственно, а затем в виде свойств в новом объекте `entry`.

По завершении цикла возвратите десериализованный объект `data`.

После этого добавьте следующий код в функцию `deserialize`.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},
```

```
    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Сюда будет помещен код для добавления события
        },
    // Десериализует строку запроса и возвращает
    // объект события
    "deserialize" : function(str){
            // Разбить каждую пару имя-значение на две части
            var data = str.split("&"),

            // Объявить переменные для использовании в цикле
            pairs=[], entry={}, key, val;

            // Выполнить цикл по всем парам имя-значение
            for ( x in data )
            {
                // Представим каждую пару в виде массива
                pairs = data[x].split("=");

                // Первый элемент -- это имя
                key = pairs[0];

                // Второй элемент -- это значение
                val = pairs[1];

                // Сохранить каждое значение в виде
                // свойства объекта
                entry[key] = val;
            }
            return entry;
        }

};
```

## Декодирование данных, отправляемых в закодированном виде вместе с URL

Прежде чем функция `fx.deserialize` будет действительно готова к использованию, ее необходимо несколько видоизменить, чтобы обеспечить декодирование данных, отправляемых в закодированном виде вместе с URL. Дело в том, что в процессе сериализации данных строки кодируются таким образом, чтобы их можно было передавать в строке запроса. Это означает, что строка "I'm testing & logging" в результате сериализации будет преобразована в следующую строку.

```
I'm+testing+%26+logging!
```

Чтобы обратить этот процесс, следует заменить все знаки "плюс"(+) пробелами с помощью регулярного выражения `/\+/g`; этому выражению соответствуют только знаки `+`. Символ `g`, следующий за закрывающим разделителем выражения, делает поиск, осуществляемый данным регулярным выражением, глобальным, поэтому будет заменено более чем одно совпадение.

Затем следует использовать функцию JavaScript `decodeURIComponent()`. Создайте в `fx` новую функцию `urldecode` и введите для нее следующий код.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Сюда будет помещен код для добавления события
        },
    // Десериализует строку запроса и возвращает объект события
    "deserialize" : function(str){
            // Разбить каждую пару "имя-значение" на две части
            var data = str.split("&"),

            // Объявить переменные для использовании в цикле
            pairs=[], entry={}, key, val;

            // Выполнить цикл по всем парам "имя-значение"
            for ( x in data )
            {
                // Представим каждую пару в виде массива
                pairs = data[x].split("=");

                // Первый элемент -- это имя
                key = pairs[0];

                // Второй элемент -- это значение
                val = pairs[1];

                // Сохранить каждое значение в виде
                // свойства объекта
                entry[key] = val;
            }
            return entry;
        },

    // Декодирует значение строки запроса
    "urldecode" : function(str) {
            // Преобразовать знаки + в пробелы
            var converted = str.replace(/\+/g, ' ');

            // Выполнить обратное преобразование остальных
            // закодированных объектов
            return decodeURIComponent(converted);
        }

};
```

Затем вставьте вызов функции `fx.urldecode` в `fx.desrialize`, добавив следующий код, выделенный полужирным шрифтом.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
```

```
    "addevent" : function(data, formData){
            // Сюда будет помещен код для добавления события
        },
    // Десериализует строку запроса и возвращает объект события
    "deserialize" : function(str){
            // Разбить каждую пару "имя-значение" на две части
            var data = str.split("&"),

            // Объявить переменные для использовании в цикле
            pairs=[], entry={}, key, val;

            // Выполнить цикл по всем парам "имя-значение"
            for ( x in data )
            {
                // Представим каждую пару в виде массива
                pairs = data[x].split("=");

                // Первый элемент -- это имя
                key = pairs[0];

                // Второй элемент -- это значение
                val = pairs[1];

                // Обратить URL-кодирование и сохранить каждое
                // значение в виде свойства объекта
                entry[key] = fx.urldecode(val);
            }
            return entry;
        },

     "urldecode" : function(str) {...}
};
```

### Объединение данных в единый объект

Подготовив функции `fx.deserialize` и `fx.urldecode`, можно видоизменить функцию `fx.addevent`, добавив переменную (`entry`) для хранения десериализованных данных.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Преобразовать строку запроса в объект
            var entry = fx.deserialize(formData);
        },
    "deserialize" : function(str){...},
    "urldecode" : function(str...}

};
```

# Создание объектов Date

Поскольку в календарь должны добавляться лишь события, созданные для отображаемого месяца, необходимо определить отображаемые месяц и год, а также месяц и год, на которые приходится событие.

**Примечание.** На данном этапе воспользуемся встроенным объектом Date JavaScript, методы которого упрощают многие операции, связанные с датой и временем. Чтобы ознакомиться с полным описанием методов объекта Date, посетите веб-страницу по следующему адресу:
http://w3schools.com/jsref/jsref_obj_date.asp

## Добавление идентификатора в класс Calendar

Чтобы сгенерировать объект Date для текущего отображаемого месяца, необходимо добавить идентификатор (ID) в элемент h2, который отображает месяц над календарем. Для обеспечения совместимости с браузерами внесите в метод buildCalendar() класса Calendar изменения, выделенные ниже полужирным шрифтом.

```
public function buildCalendar()
{
    /*
     * Определить месяц календаря и создать массив сокращенных
     * обозначений дней недели, которые будут использованы
     * в заголовках столбцов
     */
    $cal_month = date('F Y', strtotime($this->_useDate));
    $cal_id = date('Y-m', strtotime($this->_useDate));
    $weekdays = array('Sun', 'Mon', 'Tue',
            'Wed', 'Thu', 'Fri', 'Sat');

    /*
     * Добавить заголовок в HTML-разметку календаря
     */
    $html = "\n\t<h2 id=\"month-$cal_id\">$cal_month</h2>";
    for ( $d=0, $labels=NULL; $d<7; ++$d )
    {
        $labels .= "\n\t\t<li>" . $weekdays[$d] . "</li>";
    }
    $html .= "\n\t<ul class=\"weekdays\">"
        . $labels . "\n\t</ul>";
    // Для краткости остальная часть метода опущена
)
```

**Примечание.** Использование префикса month в идентификаторе означает, что вы придерживаетесь стандартов W3, согласно которым идентификаторы элементов должны начинаться с буквы.

## Создание объектов Date в JavaScript

Для гарантии того, что новые события приходятся на текущий месяц, создайте два пустых объекта Date: для текущего месяца и нового события.

Чтобы установить значение текущего месяца для объекта Date, извлеките атрибут ID из элемента h2 с помощью метода attr(), разбейте его на подстроки в местах, где встречаются дефисы, и сохраните в переменной cdata.

Для нового события следует разбить значение `entry.event_start` на подстроки в местах, где встречаются пробелы, взять первый элемент массива (представляющий дату в формате ГГГГ-ММ-ДД) и сохранить его в переменной `date`. Далее следует разбить данные на подстроки в местах, где встречаются дефисы, и сохранить массив в переменной `edata`.

Чтобы установить свойства объектов `Date`, используйте данные из `cdata` и `edata` для установки дат в объектах `cal` и `event` соответственно.

Наконец, внесите изменения в функцию `fx.addevent` с помощью кода, выделенного ниже полужирным шрифтом.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Преобразовать строку запроса в объект
            var entry = fx.deserialize(formData),

            // Создать объект "date" для текущего месяца
            cal = new Date(NaN),

            // Создать объект "date" для нового события
            event = new Date(NaN),

            // Извлечь календарный месяц из ID H2
            cdata = $("h2").attr("id").split('-'),

            // Извлечь день, месяц и год события
            date = entry.event_start.split(' ')[0],

            // Разбить данные события на отдельные части
            edata = date.split('-');

            // Задать дату для объекта "date" календаря
            cal.setFullYear(cdata[1], cdata[2], 1);

            // Задать дату для объекта "date" события
            event.setFullYear(edata[0], edata[1], edata[2]);
        },
    "deserialize" : function(str){...},
    "urldecode" : function(str) {...}
};
```

### Устранение несогласованности часовых поясов

Поскольку время и часовой пояс не передаются в объект `Date`, в этом объекте по умолчанию устанавливается полночь по Гринвичу (`00:00:00 GMT`). Это может привести к тому, что для пользователей, находящихся в разных часовых поясах, ваши даты могут вести себя непредсказуемым образом. Чтобы решить эту проблему, необходимо учесть в датах сдвиг во времени при переходе от одного часового пояса к другому, воспользовавшись для этого двумя встроенными методами объекта `Date`: `.setMinutes()` и `.getTimezoneOffset()`.

Возвращаемое значение метода `.getTimezoneOffset()` представляет собой разницу в минутах между GMT и часовым поясом пользователя. Например, для часового пояса MST[2] (`-0700`) метод `.getTimezoneOffset()` возвратит значение `420`.

Используя метод `.setMinutes()`, можно добавить сдвиг часового пояса в объект `Date`, что возвратит дату к полуночи заданного дня, независимо от того, в каком часовом поясе находится пользователь.

Для учета описанной коррекции используйте код, выделенный ниже полужирным шрифтом.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Преобразовать строку запроса в объект
            var entry = fx.deserialize(formData),

            // Создать объект "date" для текущего месяца
            cal = new Date(NaN),

            // Создать объект "date" для нового события
            event = new Date(NaN),

            // Извлечь календарный месяц из ID H2
            cdata = $("h2").attr("id").split('-'),

            // Извлечь день, месяц и год события
            date = entry.event_start.split(' ')[0],

            // Разбить данные события на отдельные части
            edata = date.split('-');

            // Задать дату для объекта "date" календаря
            cal.setFullYear(cdata[1], cdata[2], 1);

            // Задать дату для объекта "date" события
            event.setFullYear(edata[0], edata[1], edata[2]);

            // Поскольку объект "date" создан с использованием GMT, а затем
            // вводится поправка на местный часовой пояс, скорректировать
            //  смещение, чтобы дата имела правильное значение
            event.setMinutes(event.getTimezoneOffset());
        },
    "deserialize" : function(str){...},
    "urldecode" : function(str) {...}
};
```

[2] Mountain Standard Time — *стандартное горное время* (-7 часов от Гринвича, горные штаты США). — *Примеч. ред.*

### Проверка того, что событие приходится на текущий месяц

Следующий шаг состоит в использовании условного оператора, гарантирующего, что к календарю будут присоединяться лишь те события, которые связаны с данным месяцем. Если и год, и месяц текущего календаря согласуются с датой события, можно извлечь день месяца, используя для этого метод `getDay()` объекта `Date`. Даты, выражаемые одной цифрой, мы будем дополнять ведущим нулем, что потребует преобразования значения дня месяца в строку с помощью функции `String()`.

Для выполнения описанных действий используйте код, выделенный ниже полужирным шрифтом.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Преобразовать строку запроса в объект
            var entry = fx.deserialize(formData),

            // Создать объект "date" для текущего месяца
            cal = new Date(NaN),

            // Создать объект "date" для нового события
            event = new Date(NaN),

            // Извлечь календарный месяц из ID H2
            cdata = $("h2").attr("id").split('-'),

            // Извлечь день, месяц и год события
            date = entry.event_start.split(' ')[0],

            // Разбить данные события на отдельные части
            edata = date.split('-');

            // Задать дату для объекта "date" календаря
            cal.setFullYear(cdata[1], cdata[2], 1);

            // Задать дату для объекта "date" события
            event.setFullYear(edata[0], edata[1], edata[2]);

            // Поскольку объект "date" создан с использованием GMT, а затем
            // вводится поправка на местный часовой пояс, скорректировать
            // смещение, чтобы дата имела правильное значение
            event.setMinutes(event.getTimezoneOffset());

            // Если и год, и месяц совпадают, начать процесс
            // добавления нового события в календарь
            if ( cal.getFullYear()==event.getFullYear()
                    && cal.getMonth()==event.getMonth() )
            {
                // Получить день месяца для события
                var day = String(event.getDate());
```

```
                // Добавить предшествующий нуль к дате,
                // если она выражается одной цифрой
                day = day.length==1 ? "0"+day : day;
            }
        },
    "deserialize" : function(str){...},
    "urldecode" : function(str) {...}
};
```

## Присоединение событий к календарю

Наконец-то мы готовы к тому, чтобы присоединить событие к календарю. Для этого необходимо создать новый элемент `<a>`, скрыть его, установить значение атрибута `href` и использовать название события в качестве текста ссылки.

После этого следует установить задержку длительностью одна секунда с помощью метода `.delay(1000)` и обеспечить плавное отображение нового события.

Для реализации описанных выше действий используйте код, выделенный ниже полужирным шрифтом.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},

    // Добавляет новое событие в разметку после сохранения
    "addevent" : function(data, formData){
            // Преобразовать строку запроса в объект
            var entry = fx.deserialize(formData),

            // Создать объект "date" для текущего месяца
            cal = new Date(NaN),

            // Создать объект "date" для нового события
            event = new Date(NaN),

            // Извлечь календарный месяц из ID H2
            cdata = $("h2").attr("id").split('-'),

            // Извлечь день, месяц и год события
            date = entry.event_start.split(' ')[0],

            // Разбить данные события на отдельные части
            edata = date.split('-');

            // Задать дату для объекта "date" календаря
            cal.setFullYear(cdata[1], cdata[2], 1);

            // Задать дату для объекта "date" события
            event.setFullYear(edata[0], edata[1], edata[2]);

            // Поскольку объект "date" создан с использованием GMT, а затем
            // вводится поправка на местный часовой пояс, скорректировать
            // смещение, чтобы дата имела правильное значение
            event.setMinutes(event.getTimezoneOffset());
```

```
            // Если и год, и месяц совпадают, начать процесс
            // добавления нового события в календарь
            if ( cal.getFullYear()==event.getFullYear()
                    && cal.getMonth()==event.getMonth() )
            {
                // Получить день месяца для события
                var day = String(event.getDate());

                // Добавить предшествующий нуль к дате,
                // если она выражается одной цифрой
                day = day.length==1 ? "0"+day : day;

                // Добавить новую ссылку на дату
                $("<a>")
                    .hide()
                    .attr("href", "view.php?event_id="+data)
                    .text(entry.event_title)
                    .insertAfter($("strong:contains("+day+")"))
                    .delay(1000)
                    .fadeIn("slow");
            }
        },
    "deserialize" : function(str){...},
    "urldecode" : function(str) {...}
};
```

**Примечание.** Переменная `data` пока еще не определена. Это будет сделано в следующем разделе.

А теперь вернемся к обработчику события `click` кнопки отправки формы и модифицируем обратный вызов `success` функции `$.ajax()` для выполнения метода `fx.addevent()` с помощью следующего кода, выделенного полужирным шрифтом.

```
// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Сериализовать данные формы для использования
        // с функцией $.ajax()
        var formData = $(this).parents("form").serialize();

        // Отправить данные обрабатывающему файлу
        $.ajax({
                type: "POST",
                url: processFile,
                data: formData,
                success: function(data) {
                    // Обеспечить плавное исчезновение модального окна
                    fx.boxout();

                    // Добавить событие в календарь
                    fx.addevent(data, formData);
                },
```

```
            error: function(msg) {
                alert(msg);
            }
        });
});
```

Сохраните изменения и перезагрузите страницу `http://localhost/`. Вызовите форму для создания событий и создайте новое событие, используя следующую информацию:

- **название события** — `Addition Test`;
- **время начала** — `2010-01-09 12:00:00`;
- **время окончания** — `2010-01-09 14:00:00`;
- **описание события** — `This is a test of the dynamic addition of new events to the calendar.`

Отправка формы приведет к плавному исчезновению модального окна; секундой позже название нового события плавно отобразится в календаре в нужном месте (рис. 8.2).

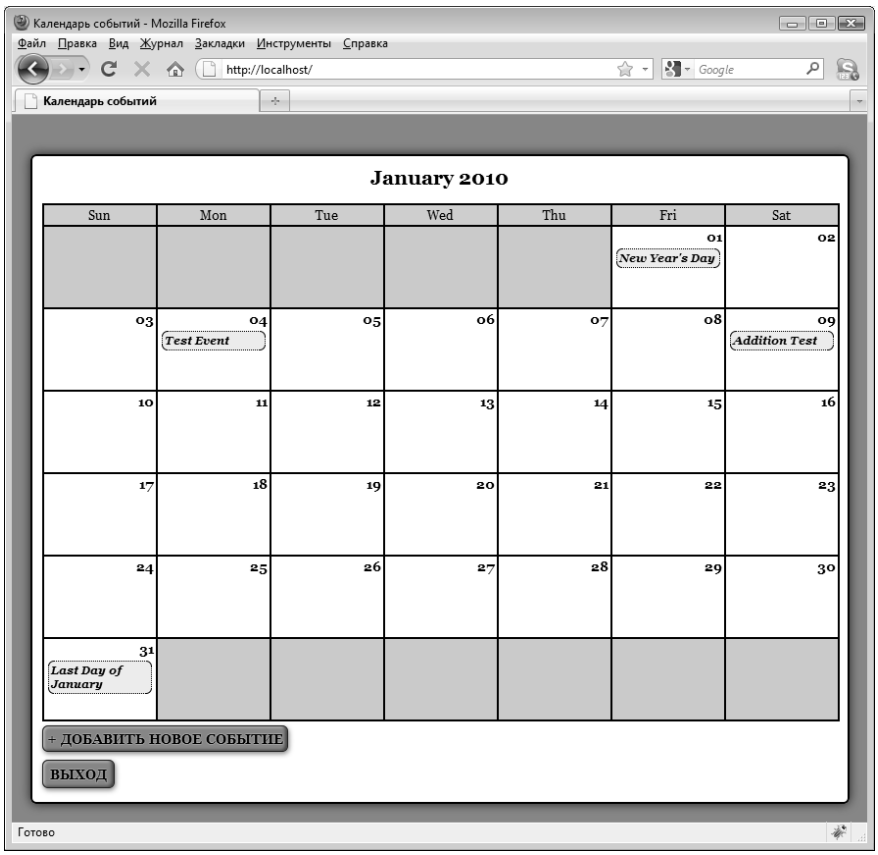



**Рис. 8.2.** Вид календаря после создания нового события

## Получение идентификатора нового события

На данном этапе новое событие нельзя просмотреть сразу после его создания; для этого необходимо обновить страницу. Поскольку идентификатор события отсутствует (после успешного добавления событий не возвращаются никакие данные), щелчок на сгенерированной ссылке приведет лишь к отображению пустого модального окна (рис. 8.3).

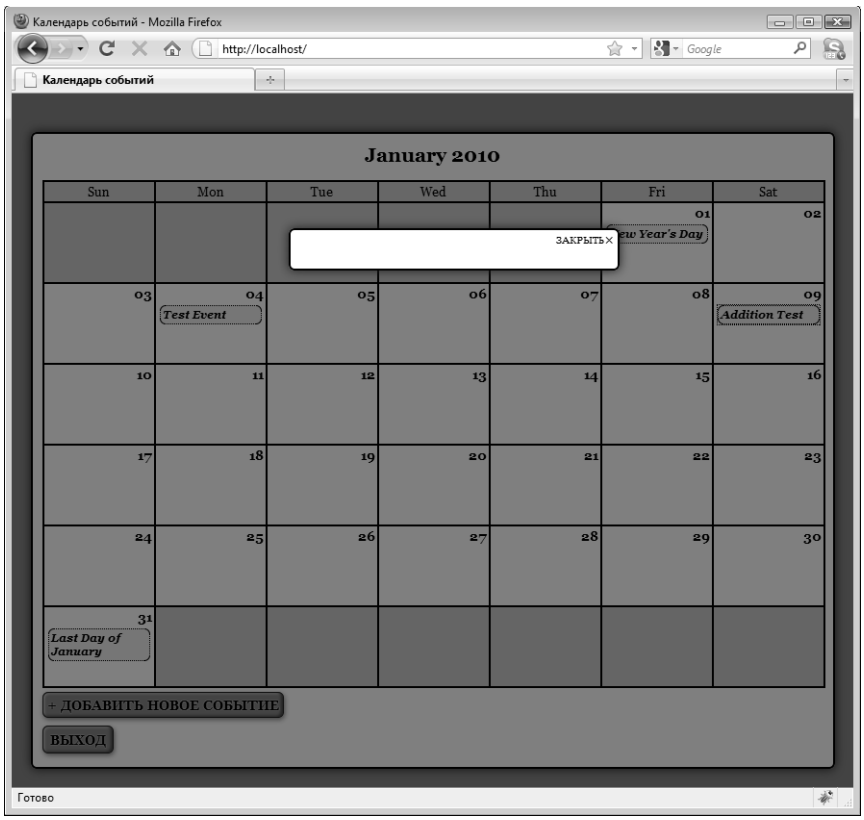



**Рис. 8.3.** Событие не могло быть загружено ввиду отсутствия его идентификатора

### *Модификация метода создания событий*

Чтобы вновь создаваемые события сразу же становились доступными для просмотра, необходимо внести небольшое изменение в класс `Calendar`. Откройте файл класса (`/sys/class/class.calendar.inc.php`) и найдите в нем метод `processForm()`.

Измените команду `return` этого метода таким образом, чтобы она выводила идентификатор (ID) последней вставленной записи, использовав для этого метод PDO `lastInserttId()`.

```
public function processForm()
{
    /*
     * Выход, если значение "action" задано неправильно
     */
```

```
if ( $_POST['action']!='event_edit' )
{
    return "Некорректная попытка вызова метода processForm";
}

/*
 * Извлечь данные из формы
 */
$title = htmlentities($_POST['event_title'], ENT_QUOTES);
$desc = htmlentities($_POST['event_description'], ENT_QUOTES);
$start = htmlentities($_POST['event_start'], ENT_QUOTES);
$end = htmlentities($_POST['event_end'], ENT_QUOTES);

/*
 * Если ID не был передан, создать новое событие
 */
if ( empty($_POST['event_id']) )
{
    $sql = "INSERT INTO `events`
                (`event_title`, `event_desc`, `event_start`,
                    `event_end`)
            VALUES
                (:title, :description, :start, :end)";
}

/*
 * Обновить событие, если оно редактировалось
 */
else
{
    /*
     * Привести ID события к целочисленному типу в интересах
     * безопасности
     */
    $id = (int) $_POST['event_id'];
    $sql = "UPDATE `events`
            SET
                `event_title`=:title,
                `event_desc`=:description,
                `event_start`=:start,
                `event_end`=:end
            WHERE `event_id`=$id";
}

/*
 * После привязки данных выполнить запрос создания или
 * редактирования события
 */
try
{
    $stmt = $this->db->prepare($sql);
    $stmt->bindParam(":title", $title, PDO::PARAM_STR);
    $stmt->bindParam(":description", $desc, PDO::PARAM_STR);
    $stmt->bindParam(":start", $start, PDO::PARAM_STR);
    $stmt->bindParam(":end", $end, PDO::PARAM_STR);
```

```
            $stmt->execute();
            $stmt->closeCursor();

            /*
             * Возвратить ID события
             */
            return $this->db->lastInsertId();
        }
        catch ( Exception $e )
        {
            return $e->getMessage();
        }
    }
```

Внеся указанные изменения, сохраните файл и перезагрузите страницу `http://localhost/` в браузере. После этого создайте новое событие, используя следующую информацию:

- **название события** — `ID Test`;
- **время начала** — `2010-01-06 12:00:00`;
- **время окончания** — `2010-01-06 16:00:00`;
- **описание события** — `This event should be immediately viewable after creation`.

После сохранения события его название появится в календаре. Щелкните на названии события, в результате чего событие загрузится в модальное окно (рис. 8.4).

# Редактирование событий в модальном окне

Чтобы с помощью модальных окон пользователи могли не только создавать, но и редактировать события, требуются совсем небольшие изменения. Обработчик события `click`, загружающий форму для создания событий, также легко переделывается для редактирования событий.

Сначала расширим селектор таким образом, чтобы отбирались любые элементы с классом `admin`. Это делается с помощью следующего кода, выделенного полужирным шрифтом.

```
// Отображает форму для редактирования в виде модального окна
$(".admin-options form,.admin").live("click", function(event){

        // Предотвратить отправку формы
        event.preventDefault();

        // Загрузить атрибут "action" для обрабатывающего файла
        var action = "edit_event";

        // Загрузить форму для редактирования событий и отобразить ее
        $.ajax({
                type: "POST",
                url: processFile,
                data: "action="+action,
                success: function(data){
                        // Скрыть форму
                        var form = $(data).hide(),
```

```
                    // Убедиться в существовании модального окна
                    modal = fx.initModal();

                    // Вызвать функцию boxin для создания модального оверлея
                    // и обеспечить его плавное появление
                    fx.boxin(null, modal);

                    // Загрузить форму в окно, обеспечить плавное появление
                    // содержимого и добавить класс в форму
                    form
                        .appendTo(modal)
                        .addClass("edit-form")
                        .fadeIn("slow");
                },
                error: function(msg){
                    alert(msg);
                }
            });

});
```

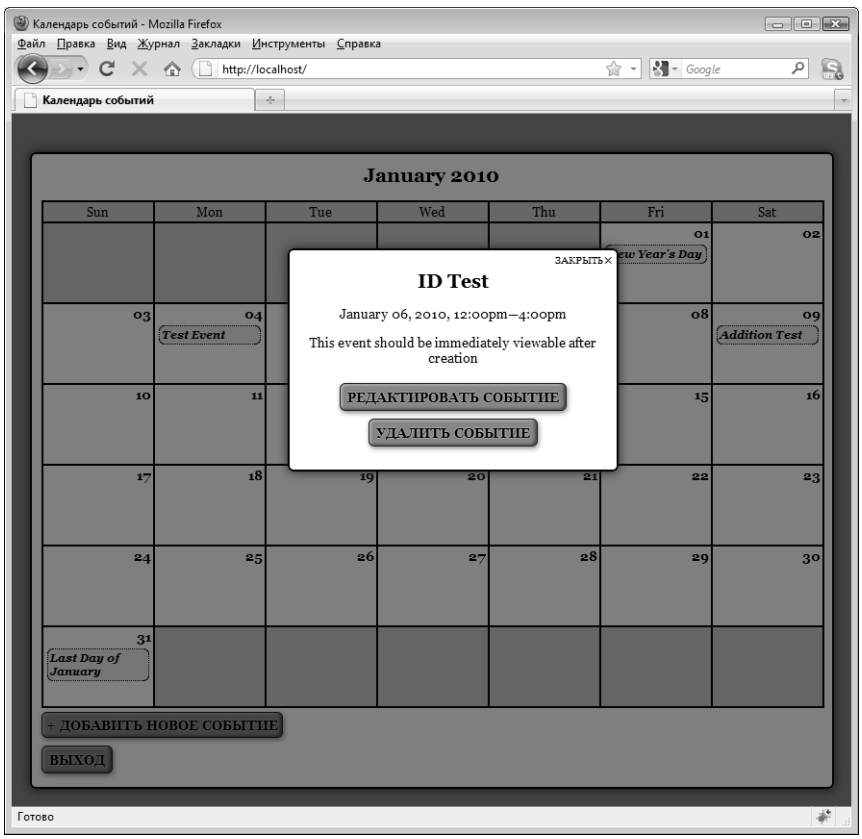



**Рис. 8.4.** Немедленная загрузка события сразу же после его создания

## Определение атрибута `action` для формы

В элементах управления редактированием, отображаемых для индивидуальных событий, названия кнопок соответствуют значениям атрибута `action`, определяющим вид действия, которое должно быть выполнено при щелчке на кнопке (например, `edit_event` для кнопки Редактировать событие и `delete_event` для кнопки Удалить событие). Эти имена и будут использоваться файлом `ajax.inc.php` в качестве значений атрибута `action` при отправке формы.

Поскольку у кнопки создания нового события отсутствует имя, мы используем для нее значение по умолчанию (`edit_event`).

Чтобы получить доступ к имени кнопки, на которой был выполнен щелчок, используем свойство `target` объекта `event`. Это свойство содержит ссылку на элемент, запустивший данное событие. Для выбора свойства `target` объекта `event` используем функцию `jQuery()`, а для извлечения имени элемента, запустившего событие, — метод `.attr()`.

Внесите в обработчик событий необходимые изменения, используя следующий код, выделенный полужирным шрифтом.

```
// Отображает форму для редактирования в виде модального окна
$(".admin-options form,.admin").live("click", function(event){

        // Предотвратить отправку формы
        event.preventDefault();

        // Задать атрибут "action" для отправки формы
        var action = $(event.target).attr("name") || "edit_event";


        // Загрузить атрибут "action" для обрабатывающего файла
        var action = "edit_event";

        // Загрузить форму для редактирования событий и отобразить ее
        $.ajax({
                type: "POST",
                url: processFile,
                data: "action="+action,
                success: function(data){
                        // Скрыть форму
                        var form = $(data).hide(),

                        // Убедиться в существовании модального окна
                        modal = fx.initModal();

                        // Вызвать функцию boxin для создания
                        // модального оверлея и обеспечить его
                        // плавное появление
                        fx.boxin(null, modal);

                        // Загрузить форму в окно, обеспечить
                        // плавное появление содержимого
                        // и добавить класс в форму
                        form
                            .appendTo(modal)
                            .addClass("edit-form")
```

```
                        .fadeIn("slow");
            },
            error: function(msg){
                alert(msg);
            }
        });

});
```

## Сохранение идентификатора события, если таковой существует

Следующее, что необходимо сделать, — это извлечь идентификатор события, если только он существует. Чтобы найти его, вновь воспользуемся свойством `event.target`, но на этот раз выполним поиск сестринского элемента с именем `event_id`, а затем сохраним его значение в переменной `id`.

Для этого внесите в обработчик событий следующие изменения, выделенные полужирным шрифтом.

```
// Отображает форму для редактирования в виде модального окна
$(".admin-options form,.admin").live("click", function(event){

        // Предотвратить отправку формы
        event.preventDefault();

        // Задать атрибут "action" для отправки формы
        var action = $(event.target).attr("name") || "edit_event";

        // Сохранить значение элемента ввода с именем event_id
        id = $(event.target)
                .siblings("input[name=event_id]")
                .val();

        // Загрузить атрибут "action" для обрабатывающего файла
        var action = "edit_event";

        // Загрузить форму для редактирования событий и отобразить ее
        $.ajax({
                type: "POST",
                url: processFile,
                data: "action="+action,
                success: function(data){
                        // Скрыть форму
                        var form = $(data).hide(),

                        // Убедиться в существовании модального окна
                        modal = fx.initModal();

                        // Вызвать функцию boxin для создания
                        // модального оверлея и обеспечить его
                        // плавное появление
                        fx.boxin(null, modal);
```

```
                    // Загрузить форму в окно, обеспечить
                    // плавное появление содержимого
                    // и добавить класс в форму
                    form
                        .appendTo(modal)
                        .addClass("edit-form")
                        .fadeIn("slow");
                },
                error: function(msg){
                    alert(msg);
                }
            });

});
```

### Добавление идентификатора события в строку запроса

Сохранив идентификатор события в переменной `id`, можно присоединить его значение к строке запроса для отправки файлу `ajax.inc.php`.

Предварительно убедившись в том, что идентификатор события установлен, создайте для элемента ввода `event_id` пару *имя-значение*. После этого присоедините данные к строке запроса, используя следующий код, выделенный полужирным шрифтом.

```
// Отображает форму для редактирования в виде модального окна
$(".admin-options form,.admin").live("click", function(event){

        // Предотвратить отправку формы
        event.preventDefault();

        // Задать атрибут "action" для отправки формы
        var action = $(event.target).attr("name") || "edit_event";

        // Сохранить значение элемента ввода с именем event_id
        id = $(event.target)
                 .siblings("input[name=event_id]")
                 .val();

        // Создать дополнительный параметр для идентификатора,
        // если он установлен
        id = ( id!=undefined ) ? "&event_id="+id : "";

        // Загрузить атрибут "action" для обрабатывающего файла
        var action = "edit_event";

        // Загрузить форму для редактирования событий и отобразить ее
        $.ajax({
                type: "POST",
                url: processFile,
                data: "action="+action+id,
                success: function(data){
                        // Скрыть форму
                        var form = $(data).hide(),
```

```
                    // Убедиться в существовании модального окна
                    modal = fx.initModal();

                    // Вызвать функцию boxin для создания
                    // модального оверлея и обеспечить его
                    // плавное появление
                    fx.boxin(null, modal);

                    // Загрузить форму в окно, обеспечить
                    // плавное появление содержимого
                    // и добавить класс в форму
                    form
                        .appendTo(modal)
                        .addClass("edit-form")
                        .fadeIn("slow");
                },
                error: function(msg){
                    alert(msg);
                }
            });

    });
```

## Удаление информации о событии из модального окна

Чтобы заменить содержимое модального окна формой для редактирования событий, необходимо сначала удалить отображаемую информацию о событии.

В том фрагменте кода обработчика `success`, где вызывается метод `fx.iniModal()`, выберите все дочерние элементы, отличные от кнопки Закрыть, и удалите их. После этого вызовите метод `.end()`, чтобы вернуться к первоначально выбранному набору элементов модального окна. (После вызова метода `.children()` объект jQuery будет ссылаться только на дочерние элементы, которые вы только что удалили.)

Для этого добавьте следующий код, выделенный полужирным шрифтом.

```
// Отображает форму для редактирования в виде модального окна
$(".admin-options form,.admin").live("click", function(event){

        // Предотвратить отправку формы
        event.preventDefault();

        // Задать атрибут "action" для отправки формы
        var action = $(event.target).attr("name") || "edit_event";

        // Сохранить значение элемента ввода с именем event_id
        id = $(event.target)
                .siblings("input[name=event_id]")
                .val();

        // Создать дополнительный параметр для идентификатора,
        // если он установлен
        id = ( id!=undefined ) ? "&event_id="+id : "";

        // Загрузить атрибут "action" для обрабатывающего файла
        var action = "edit_event";
```

```
        // Загрузить форму для редактирования событий
        // и отобразить ее
        $.ajax({
                type: "POST",
                url: processFile,
                data: "action="+action+id,
                success: function(data){
                        // Скрыть форму
                        var form = $(data).hide(),

                        // Убедиться в существовании модального окна
                        modal = fx.initModal();
                             .children(":not(.modal-close-btn)")
                             .remove()
                              .end();

                        // Вызвать функцию boxin для создания
                        // модального оверлея и обеспечить его
                        // плавное появление
                        fx.boxin(null, modal);

                        // Загрузить форму в окно, обеспечить
                        // плавное появление содержимого
                        // и добавить класс в форму
                        form
                            .appendTo(modal)
                            .addClass("edit-form")
                            .fadeIn("slow");
                },
                error: function(msg){
                    alert(msg);
                }
            });

    });
```

Сохраните этот файл, перезагрузите страницу `http://localhost/` в своем браузере и щелкните на названии события *New Year's Day*, чтобы отобразить его описание. Щелкните на кнопке Редактировать событие. Описание события исчезнет, и вместо него плавно отобразится форма для редактирования событий с загруженными в нее данными (рис. 8.5).

## Добавление только новых событий в календарь

Если отредактировать событие *New Year's Day*, а затем сохранить его, то в соответствующей ячейке календаря отобразятся два события (рис. 8.6).

Чтобы этого не происходило, необходимо внести некоторые изменения в обработку события `click` при отправке формы. Поскольку идентификаторы (ID) редактируемых событий загружаются в скрытый элемент ввода `event_id` формы редактирования, можно проверить длину значения, находящегося в этом элементе ввода. Если она не равна нулю, то вызывать функцию `fx.addevent()` не следует.

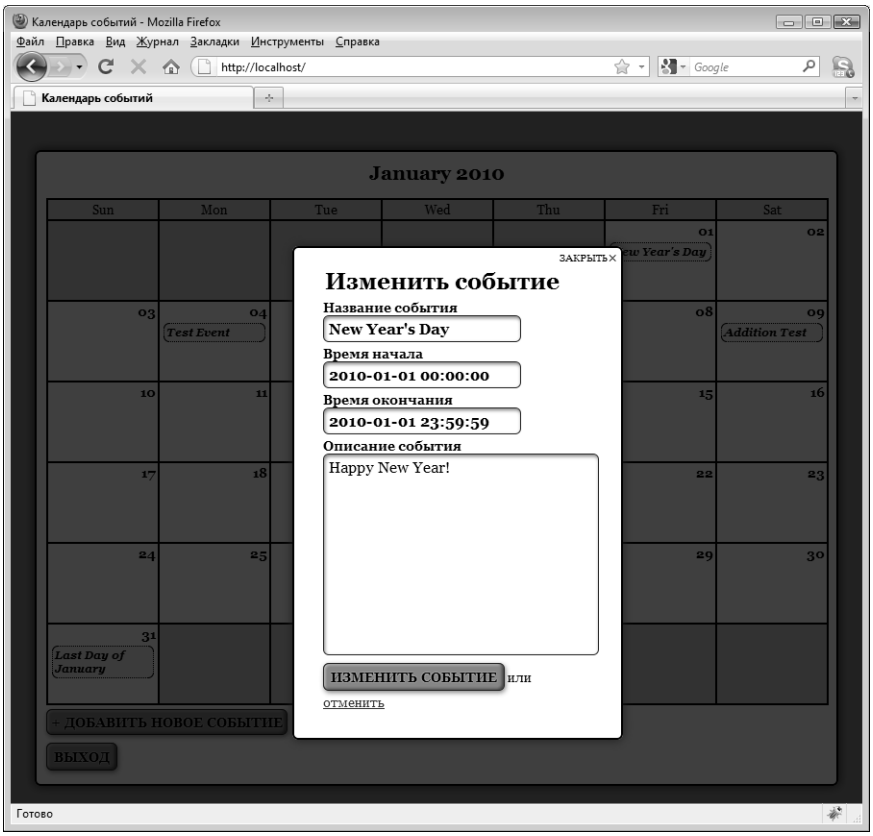



**Рис. 8.5.** Редактирование события в модальном окне

Для выполнения этой проверки вставьте следующий код, выделенный полужирным шрифтом.

```
// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Сериализовать данные формы для использования с $.ajax()
        var formData = $(this).parents("form").serialize();

        // Отправить данные обрабатывающему файлу
        $.ajax({
                type: "POST",
                url: processFile,
                data: formData,
                success: function(data) {
                    // Обеспечить плавное исчезновение модального окна
                    fx.boxout();
```

```
                // Если это новое событие, добавить
                // его в календарь
                if ( $("[name=event_id]").val().length==0 )
                {
                    fx.addevent(data, formData);
                }
            },
            error: function(msg) {
                alert(msg);
            }
        });
});
```

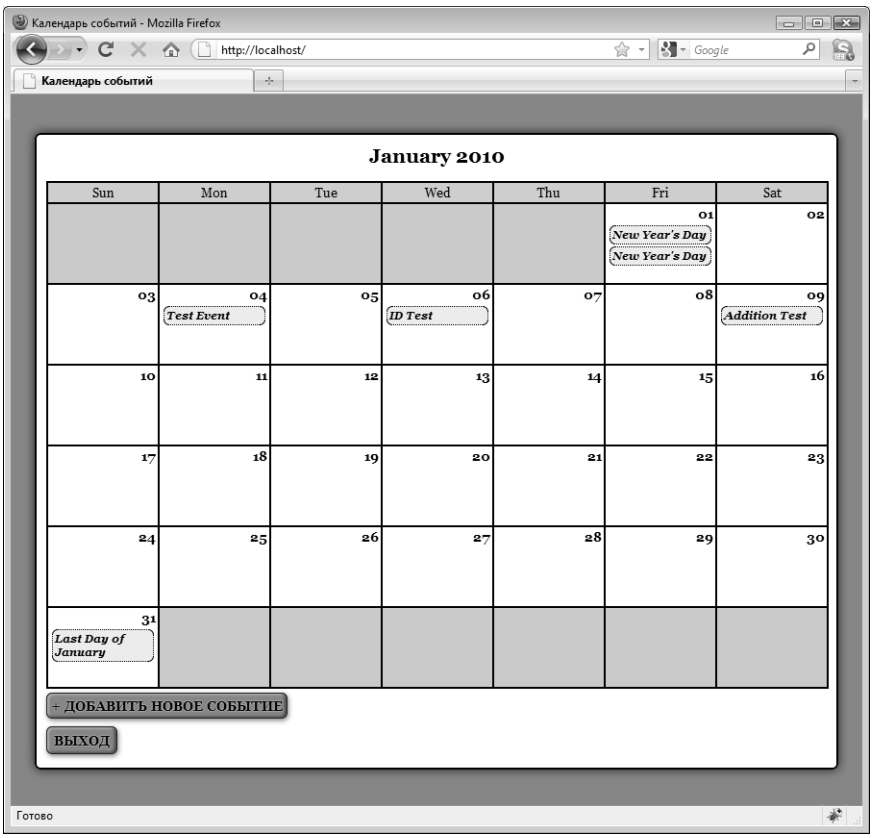



**Рис. 8.6.** Дублирование названия отредактированного события в календаре после его сохранения

После внесенных изменений процесс редактирования событий не будет сопровождаться появлением ложных дубликатов названий событий, которые могут сбивать пользователей с толку.

# Подтверждение удаления событий в модальном окне

Нам осталось убрать в приложении еще одну шероховатость и сделать так, чтобы и удаление событий можно было выполнять без обновления страницы. Почти все, что для этого нужно, уже имеется в сценарии, поэтому добавление этой функциональности потребует внесения лишь незначительных изменений в существующий код.

## Отображение окна подтверждения

Для отображения диалогового окна, предоставляющего пользователю возможность подтвердить свои намерения после щелчка на кнопке Удалить событие, необходимо включить в поисковый массив в файле `ajax.inc.php` дополнительный элемент, как показано ниже.

```
/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_view' => array(
                'object' => 'Calendar',
                'method' => 'displayEvent'
            ),
        'edit_event' => array(
                'object' => 'Calendar',
                'method' => 'displayForm'
            ),
        'event_edit' => array(
                'object' => 'Calendar',
                'method' => 'processForm'
            ),
        'delete_event' => array(
                'object' => 'Calendar',
                'method' => 'confirmDelete'
```

```
            )
    );

/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);

    /*
     * Проверить наличие идентификатора ID и выполнить
     * необходимую коррекцию
     */
    if ( isset($_POST['event_id']) )
    {
        $id = (int) $_POST['event_id'];
    }
    else { $id = NULL; }

    echo $obj->$use_array['method']($id);
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}
```

Теперь при щелчке на кнопке Удалить событие модального окна пользователю будет предложено подтвердить выполнение этой операции (рис. 8.7).

## Настройка обработчика события отправки формы, предназначенной для удаления события

Подтверждение удаления события требует дополнительной модификации файла `init.js`. Чтобы все работало правильно, следует сохранить значение отправки формы и передать его в обрабатывающий файл. Это необходимо сделать потому, что форма может быть передана либо со значением Да, удалить, либо со значением Нет! Это была шутка!, и сценарий проверяет, на какой кнопке был выполнен щелчок, чтобы определить, какое действие следует предпринять.

Для получения значения кнопки используйте ключевое слово `this` в качестве селектора jQuery, а затем сохраните строку, возвращенную из `.val()`, в переменной `submitVal`. После этого проверьте, является ли `confirm_delete` значением атрибута `name` кнопки. Если это так, присоедините действие `confirm_delete` и значение кнопки к строке запроса перед ее отправкой.

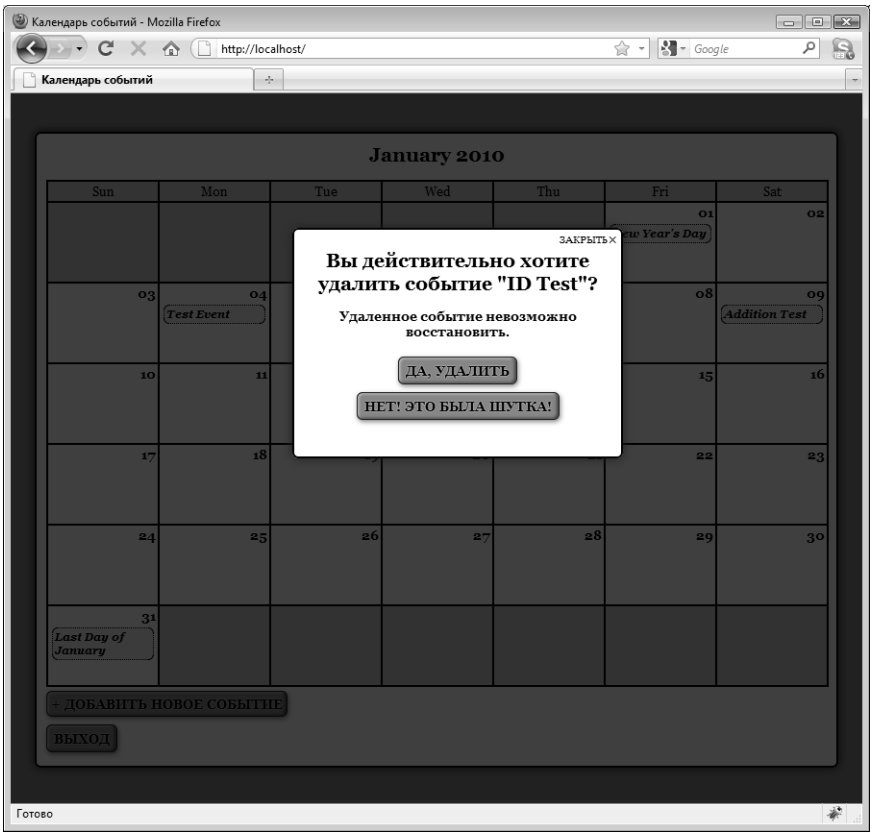



**Рис. 8.7.** Подтверждение удаления события в модальном окне

Чтобы это реализовать, вставьте следующий код, выделенный полужирным шрифтом.

```
// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Сериализовать данные формы для использования
        // с функцией $.ajax()
        var formData = $(this).parents("form").serialize(),

        // Сохранить значение кнопки "submit"
            submitVal = $(this).val();

        // Если это форма для удаления, присоединить действие
        if ( $(this).attr("name")=="confirm_delete" )
        {

            // Добавить необходимую информацию в строку запроса
```

```
            formData += "&action=confirm_delete"
                + "&confirm_delete="+submitVal;
        }

        // Отправить данные обрабатывающему файлу
        $.ajax({
                type: "POST",
                url: processFile,
                data: formData,
                success: function(data) {
                    // Обеспечить плавное исчезновение модального окна
                    fx.boxout();

                    // Если это новое событие, добавить
                    // его в календарь
                    if ( $("[name=event_id]").val().length==0 )
                    {
                        fx.addevent(data, formData);
                    }
                },
                error: function(msg) {
                    alert(msg);
                }
            });
    });
```

### *Модификация обрабатывающего файла*

Наконец, чтобы кнопка удаления события заработала, в поисковый массив в файле `ajax.inc.php` необходимо добавить дополнительный элемент.

```
/*
 * Запуск сеанса
 */
session_start();

/*
 * Включить необходимые файлы
 */
include_once '../../../sys/config/db-cred.inc.php';

/*
 * Определить константы для конфигурационной информации
 */
foreach ( $C as $name => $val )
{
    define($name, $val);
}

/*
 * Создать поисковый массив для действий, выполняемых над формой
 */
$actions = array(
        'event_view' => array(
```

```
                'object' => 'Calendar',
                'method' => 'displayEvent'
            ),
        'edit_event' => array(
                'object' => 'Calendar',
                'method' => 'displayForm'
            ),
        'event_edit' => array(
                'object' => 'Calendar',
                'method' => 'processForm'
            ),
        'delete_event' => array(
                'object' => 'Calendar',
                'method' => 'confirmDelete'
            ),
        'confirm_delete' => array(
                'object' => 'Calendar',
                'method' => 'confirmDelete'
            )
    );

/*
 * Убедиться в том, что маркер защиты от CSRF был передан и что
 * запрошенное действие существует в поисковом массиве
 */
if ( isset($actions[$_POST['action']]) )
{
    $use_array = $actions[$_POST['action']];
    $obj = new $use_array['object']($dbo);

    /*
     * Проверить наличие идентификатора ID и выполнить
     * необходимую коррекцию
     */
    if ( isset($_POST['event_id']) )
    {
        $id = (int) $_POST['event_id'];
    }
    else { $id = NULL; }

    echo $obj->$use_array['method']($id);
}

function __autoload($class_name)
{
    $filename = '../../../sys/class/class.'
        . strtolower($class_name) . '.inc.php';
    if ( file_exists($filename) )
    {
        include_once $filename;
    }
}
```

Можете протестировать предыдущий код, удалив из календаря событие *ID Test*. После закрытия модального окна событие все еще будет отображаться и реагировать на щелчки. Однако если вы попытаетесь просмотреть подробную информацию о событии, то она окажется недоступной (рис. 8.8).

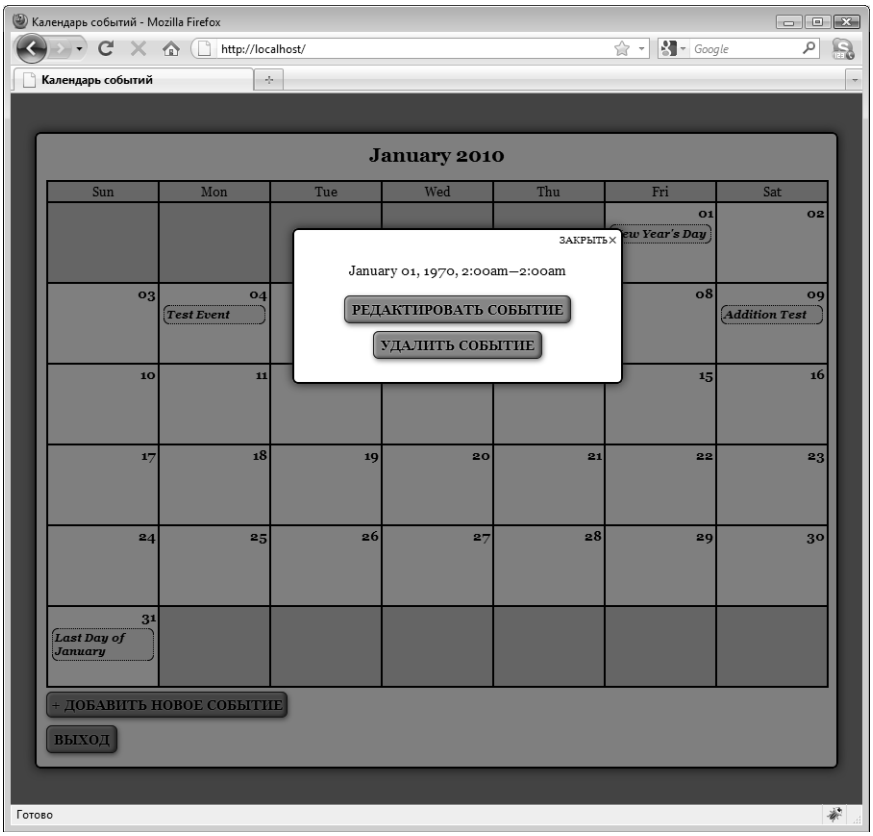



**Рис. 8.8.** Поскольку событие больше не существует, выводимые для него данные не имеют смысла

## Исключение события из календаря после его удаления

Разумеется, нельзя мириться с той неразберихой, которую вносит отображение несуществующих событий после того, как они уже удалены. Поэтому следует позаботиться о том, чтобы такие события своевременно исключались и из календаря.

Для этого добавим в объектный литерал `fx` новую функцию `removeevent`. Эта функция будет использовать класс `active`, применяемый к событиям, когда они вызываются в модальном окне, для плавного уменьшения их видимости и исключения их из DOM. Добавьте эту функцию с помощью следующего кода, выделенного полужирным шрифтом.

```
fx = {
    "initModal" : function() {...},
    "boxin" : function(data, modal) {...},
    "boxout" : function(event) {...},
    "addevent" : function(data, formData) {...},
```

```
        // Исключает событие из разметки после удаления
        "removeevent" : function()
        {
                // Исключить любое событие с классом "active"
                $(".active")
                    .fadeOut("slow", function(){
                            $(this).remove();
                        });
            },

        "deserialize" : function(str){...},
        "urldecode" : function(str) {...}
    };
```

### Модификация обработчика события отправки формы для исключения удаленных событий

Чтобы исключить из календаря события после того, как они удалены, добавьте в обработчик события отправки формы новую переменную `remove`. В этой переменной будет храниться булево значение, указывающее сценарию, подлежит ли событие исключению. По умолчанию для этой переменной устанавливается значение `false`, которое говорит о том, что событие не следует исключать из календаря.

Единственным условием, при котором событие подлежит исключению из календаря, является выполнение щелчка на кнопке Да, удалить. Добавьте проверку того, что текст `"Да, удалить"` действительно является значением кнопки отправки формы, и, если это действительно так, установите для переменной `remove` значение `true`.

В обработчике события успешного выполнения поместите условный оператор, который проверяет значение `remove` и, если оно равно `true`, запускает метод `fx.removeevent()`.

Наконец, дабы не допустить включения в календарь пустых элементов, модифицируйте условный оператор, который запускает функцию `fx.addevent()`, таким образом, чтобы перед выполнением осуществлялась проверка того, что значение `remove` равно `false`.

Для реализации этого внесите в код следующие изменения, выделенные полужирным шрифтом.

```
// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Сериализовать данные формы для использования
        // с функцией $.ajax()
        var formData = $(this).parents("form").serialize(),

        // Сохранить значение кнопки "submit"
        submitVal = $(this).val(),

        // Определить, подлежит ли событие исключению из календаря
        remove = false;
```

```
        // Если это форма для удаления, присоединить действие
        if ( $(this).attr("name")=="confirm_delete" )
        {

            // Добавить необходимую информацию в строку запроса
            formData += "&action=confirm_delete"
                + "&confirm_delete="+submitVal;

            // Если событие действительно удаляется, установить
            // флаг для исключения его из разметки
            if ( submitVal=="Да, удалить" )
            {
                remove = true;
            }
        }

        // Отправить данные обрабатывающему файлу
        $.ajax({
                type: "POST",
                url: processFile,
                data: formData,
                success: function(data) {
                    // Если это событие удалено, исключить
                    // его из разметки
                    if ( remove===true )
                    {
                        fx.removeevent();
                    }
                    // Обеспечить плавное исчезновение модального окна
                    fx.boxout();

                    // Если это новое событие, добавить
                    // его в календарь
                    if ( $("[name=event_id]").val().length==0
                        && remove===false )
                    {
                        fx.addevent(data, formData);
                    }
                },
                error: function(msg) {
                    alert(msg);
                }
            });
    });
```

Сохраните изменения, перезагрузите страницу `http://localhost/` и откройте описание события *Test Event*. Удалите это событие. После щелчка на кнопке Да, удалить модальное окно и название события плавно исчезнут, и событие навсегда удалится из календаря, устраняя возможность возникновения ситуаций, сбивающих пользователя с толку (рис. 8.9).

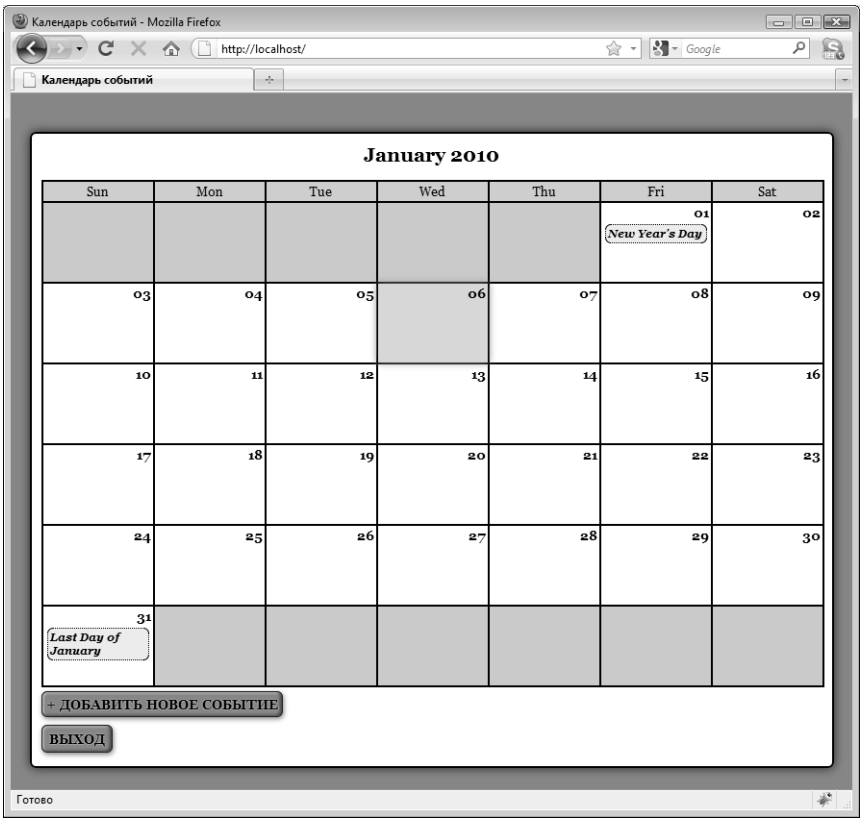



**Рис. 8.9.** После удаления события Test Event его название было исключено из календаря

# Резюме

В этой главе были реализованы элементы управления, с помощью которых пользователи могут быстро создавать, редактировать и удалять события без обновления страницы. Это позволило намного повысить эффективность приложения и дружественность его интерфейса по отношению к пользователю.

В следующей главе вы узнаете о том, как использовать регулярные выражения для проверки допустимости данных, введенных в форму при редактировании событий, чтобы предотвратить попадание в базу данных информации, которая могла бы помешать нормальной работе приложения.

# Часть IV

# Дополнительные возможности jQuery и PHP

Итак, на данном этапе у нас уже имеется приложение, реализующее основные функции календаря событий. Тем не менее мы можем сделать еще кое-что для улучшения его пользовательского интерфейса, например предусмотреть проверку данных, вводимых в форму.

Эта часть книги посвящена рассмотрению ряда сложных тем. В частности, вы узнаете о том, как организовать верификацию данных, вводимых пользователем, с помощью регулярных выражений и как создавать подключаемые модули для jQuery.

Глава 9

# Проверка форм с помощью регулярных выражений

Как разработчик, вы отвечаете за то, чтобы вводимые пользователем данные были пригодны для использования в приложении, и поэтому должны позаботиться о проверке любой критически важной информации до того, как она будет сохранена в базе данных.

Для нашего приложения такими критически важными входными данными являются даты событий: использование неправильного формата дат при их вводе может стать причиной возникновения сбоев в нескольких местах приложения. Чтобы гарантировать запись в базу данных лишь тех дат, которые соответствуют установленному формату, организуем их проверку с помощью **регулярных выражений** — мощного средства шаблонного поиска, предоставляющего в распоряжение разработчика гораздо более широкие возможности, чем поиск по простому совпадению строк.

Прежде чем приступить к добавлению элементов проверки допустимости данных в приложение, вы должны приобрести хотя бы минимальные навыки в использовании регулярных выражений. Первый раздел главы посвящен ознакомлению с базовым синтаксисом регулярных выражений. После этого регулярные выражения будут применены для проверки допустимости данных как на клиентской, так и на серверной стороне.

## Введение в регулярные выражения

Новички часто испытывают страх перед регулярными выражениями и считают, что с ними очень трудно работать. Действительно, синтаксис регулярных выражений довольно сложен и не прощает даже малейших ошибок. Однако после преодоления трудностей начального этапа обучения регулярные выражения станут невероятно мощным инструментом в ваших руках, имеющим бесчисленное множество всевозможных применений в повседневной практике программирования.

### Базовый синтаксис регулярных выражений

В данной книге вы познакомитесь с синтаксисом Perl-совместимых регулярных выражений (Perl-Compatible Regular Expressions, PCRE). Этот синтаксис совместим с PHP и JavaScript, а также с большинством других языков программирования.

**Примечание.** Более подробные сведения о PCRE можно найти по следующему адресу:
`http://ru.wikipedia.org/wiki/PCRE`

### *Создание тестового файла*

Чтобы научиться использовать регулярные выражения, вам потребуется тестовый файл. Создайте в папке `public` новый файл, назовите его `regex.php` и поместите в него следующий код.

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <title>Демонстрация регулярных выражений</title>
    <style type="text/css">
        em {
            background-color: #FF0;
            border-top: 1px solid #000;
            border-bottom: 1px solid #000;
        }
    </style>
</head>

<body>
<?php

/*
 * Строка для хранения текста, используемого в примерах
 * работы с регулярными выражениями
 */

$string = <<<TEST_DATA

<h2>Тестирование регулярных выражений</h2>
<p>
    In this document, there is a lot of text that can be matched
    using regex. The benefit of using a regular expression is much
    more flexible &mdash; albeit complex &mdash; syntax for text
    pattern matching.
</p>
<p>
    After you get the hang of regular expressions, also called
    regexes, they will become a powerful tool for pattern matching.
</p>
<hr />
TEST_DATA;

/*
 * Начнем с простого вывода данных
```

```
 */
echo $string;

?>

</body>

</html>
```

Сохранив этот файл и загрузив в браузер страницу `http://localhost/regex.php`, вы увидите следующий результат (рис. 9.1).

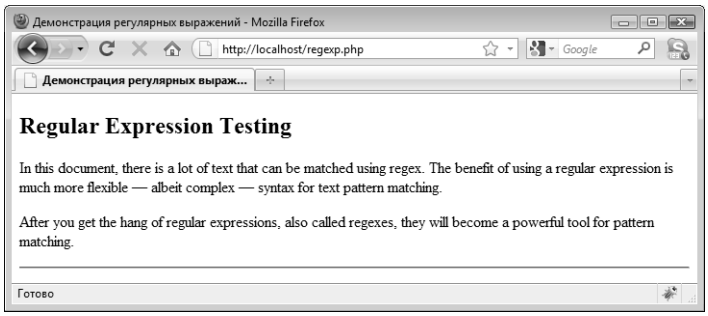



**Рис. 9.1.** Тестовый файл, используемый для апробации регулярных выражений

## *Замена текста с помощью регулярных выражений*

Для обеспечения визуального контроля результатов работы регулярных выражений будем окружать найденные элементы дескрипторами `<em>`, которые, согласно их определению в тестовом документе, устанавливают верхнюю и нижнюю линии рамки и желтый цвет фона.

В PHP замена текста реализуется с помощью функции `preg_replace()`, аналогичной функции `str_replace()`. Аргументами этой функции служат шаблон, для которого выполняется поиск соответствий; строка (или шаблон) замены, которой заменяется найденное соответствие шаблону; и наконец, строка, в пределах которой осуществляется поиск.

```
preg_replace($pattern, $replacement, $string);
```

**Примечание.** Буква *p* в `preg_replace()` означает использование PCRE. В PHP имеется также функция `ereg_replace()`, использующая несколько другой синтаксис регулярных выражений POSIX. В то же время, начиная с версии PHP 5.3.0 семейство функций `ereg` считается устаревшим и его применение не рекомендуется.

На базовом уровне единственным различием между функциями `str_replace()` и `preg_replace()` является то, что в элементе, передаваемом функции `preg_replace()` в качестве шаблона, должны использоваться *разделители*, позволяющие функции определить, какая часть регулярного выражения является шаблоном, а какая представляет *модификаторы*, или флаги, влияющие на поведение компилятора и (или) интерпретатора регулярных выражений. Более подробно о модификаторах будет сказано далее.

В функции `preg_replace()` ограничителем в шаблонах может служить любой символ, не являющийся алфавитно-цифровым символом, обратной косой чертой

или пробельным символом, который располагается в начале и в конце шаблона. Чаще всего в качестве ограничителей используют символы косой черты (/) или "решетки" (#). Например, для нахождения подстроки `cat` в строке следует использовать шаблон `/cat/` (или `#cat#`, `%cat%`, `@cat@` и т.п.).

### Сравнение методов замены с помощью строк и регулярных выражений

С целью исследования различий между функциями `str_replace()` и `preg_replace()` применим каждую из них для визуального выделения вхождений слова *regular* дескрипторами `<em>`. Внесите в файл `regexp.php` следующие изменения.

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <title>Демонстрация регулярных выражений</title>
    <style type="text/css">
        em {
            background-color: #FF0;
            border-top: 1px solid #000;
            border-bottom: 1px solid #000;
        }
    </style>
</head>

<body>
<?php

/*
 * Строка для хранения текста, используемого в примерах
 * работы с регулярными выражениями
 */

$string = <<<TEST_DATA

<h2>Тестирование регулярных выражений</h2>
<p>
    In this document, there is a lot of text that can be matched
    using regex. The benefit of using a regular expression is much
    more flexible &mdash; albeit complex &mdash; syntax for text
    pattern matching.
</p>
<p>
    After you get the hang of regular expressions, also called
    regexes, they will become a powerful tool for pattern matching.
</p>
<hr />
TEST_DATA;
```

```
/*
 * Использовать функцию str_replace() для визуального выделения
 * всех вхождений слова "regular"
 */
echo str_replace("regular", "<em>regular</em>", $string);
/*
 * Использовать функцию preg_replace() для визуального выделения
 * всех вхождений слова "regular"
 */
echo preg_replace("/regular/", "<em>regular</em>", $string);

?>

</body>

</html>
```

После выполнения этого сценария информация будет выведена дважды с идентичными результатами (рис. 9.2).

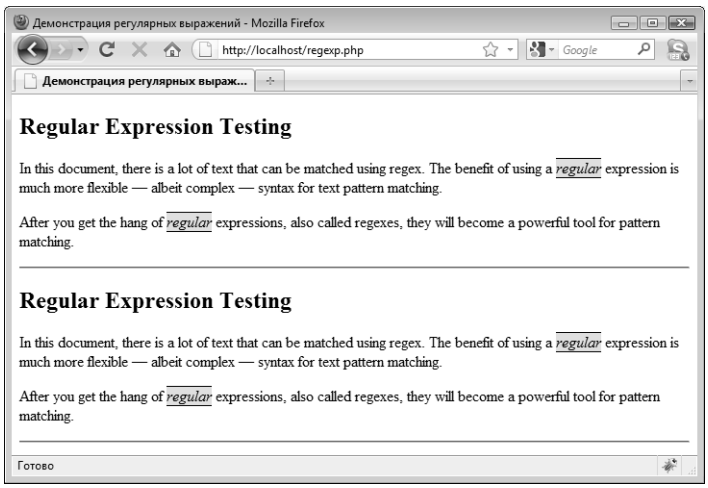



**Рис. 9.2.** Выделение слова "regular" с помощью строковой замены и регулярного выражения

## Детализация информации на основе модификаторов шаблонов

Возможно, вы заметили, что слово *regular* в заголовке не выделено. Это произошло потому, что в предыдущем примере учитывался регистр букв.

Чтобы разрешить эту проблему в случае строковой замены, можно использовать функцию `str_ireplace()`, которая почти идентична функции `str_replace()`, за исключением того, что является нечувствительной к регистру.

В случае регулярных выражений по-прежнему будет использоваться функция `preg_replace()`, но на этот раз с модификатором, устанавливающим нечувствительность к регистру, т.е. делающим различие между верхним и нижним регистра-

ми символов несущественным. Модификатор — это буква, следующая за разделителем шаблона и предоставляющая регулярному выражению дополнительную информацию относительно того, как именно должны обрабатываться шаблоны. Нечувствительность к регистру задается модификатором `i`.

Модифицируйте файл `regexp.php` так, чтобы использовались нечувствительные к регистру функции замены, внеся изменения, выделенные ниже полужирным шрифтом.

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <title>Демонстрация регулярных выражений</title>
    <style type="text/css">
        em {
            background-color: #FF0;
            border-top: 1px solid #000;
            border-bottom: 1px solid #000;
        }
    </style>
</head>

<body>
<?php

/*
 * Строка для хранения текста, используемого в примерах
 * работы с регулярными выражениями
 */

$string = <<<TEST_DATA

<h2>Тестирование регулярных выражений</h2>
<p>
    In this document, there is a lot of text that can be matched
    using regex. The benefit of using a regular expression is much
    more flexible &mdash; albeit complex &mdash; syntax for text
    pattern matching.
</p>
<p>
    After you get the hang of regular expressions, also called
    regexes, they will become a powerful tool for pattern matching.
</p>
<hr />
TEST_DATA;

/*
 * Использовать функцию str_ireplace() для визуального выделения
 * всех вхождений слова "regular"
```

```
 */
echo str_ireplace("regular", "<em>regular</em>", $string);
/*
 * Использовать функцию preg_replace() для визуального выделения
 * всех вхождений слова "regular"
 */
echo preg_replace("/regular/i", "<em>regular</em>", $string);

?>

</body>

</html>
```

Теперь загрузка файла в браузер приведет к визуальному выделению всех вхождений слова *regular* независимо от регистра букв (рис. 9.3).

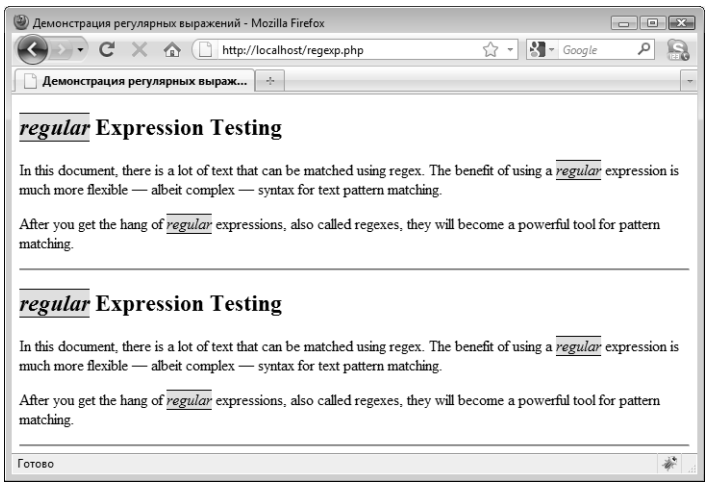



**Рис. 9.3.** Результат выполнения тестового поиска без учета регистра букв

Нетрудно заметить, что у этого подхода имеется один недостаток: прописная буква *R* в названии документа в результате замены превратилась в строчную *r*. В следующем разделе показано, как использование групп в регулярных выражениях помогает решить эту проблему.

## Использование обратных ссылок в регулярных выражениях

Истинная мощь регулярных выражений начинает проявляться тогда, когда применяются их наиболее часто используемые возможности: группирование в подшаблоны и обратные ссылки. Группа (подшаблон) — это часть шаблона, заключенная в круглые скобки. На группу можно сослаться в строке замены (или далее в шаблоне) с помощью *обратной ссылки*, которая представляет собой нумерованную ссылку на именованную группу.

Хотя это и может казаться чересчур запутанным, но на практике все получается довольно просто. Вместе с каждым набором символов, заключенным в круглые скобки, хранится его порядковый номер в регулярном выражении, отсчитываемый в на-

правлении слева направо. Для ссылки на набор используется нумерованная обратная ссылка, которой предшествует обратная косая черта (\1) или знак доллара ($1).

Обратные ссылки позволяют использовать в строке замены не только заранее заданные значения, как в случае функции str_replace() и ей подобных, но и *значения найденных соответствий шаблону*.

Чтобы в предыдущем примере подставляемое содержимое имело соответствующий регистр, функцию str_replace() нужно использовать два раза, тогда как использование обратной ссылки в функции preg_replace() позволяет добиться того же результата всего лишь однократным использованием данной функции.

Убедитесь сами, насколько мощным средством являются обратные ссылки в регулярных выражениях, внеся в файл regexp.php следующие изменения.

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <title>Демонстрация регулярных выражений</title>
    <style type="text/css">
        em {
            background-color: #FF0;
            border-top: 1px solid #000;
            border-bottom: 1px solid #000;
        }
    </style>
</head>

<body>
<?php

/*
 * Строка для хранения текста, используемого в примерах
 * работы с регулярными выражениями
 */

$string = <<<TEST_DATA

<h2>Тестирование регулярных выражений</h2>
<p>
    In this document, there is a lot of text that can be matched
    using regex. The benefit of using a regular expression is much
    more flexible &mdash; albeit complex &mdash; syntax for text
    pattern matching.
</p>
<p>
    After you get the hang of regular expressions, also called
    regexes, they will become a powerful tool for pattern matching.
</p>
<hr />
TEST_DATA;
```

```
/*
 * Использовать функцию str_replace() для визуального выделения
 * всех вхождений слова "regular"
 */
$check1 = str_replace("regular", "<em>regular</em>", $string);

/*
 * Вновь использовать функцию str_replace() для визуального выделения
 * всех вхождений слова "Regular", начинающегося с прописной буквы
 */
echo str_replace("Regular", "<em>Regular</em>", $check1);
/*
 * Использовать функцию preg_replace() для выделения всех вхождений
 * слова "regular" без учета регистра
 */
echo preg_replace("/(regular)/i", "<em>$1</em>", $string);

?>

</body>

</html>
```

Предыдущий код отлично иллюстрирует тот факт, что при любом более или менее сложном поиске по шаблонам использование функции `str_replace()` начинает доставлять неудобства. Как бы то ни было, сохранив указанные изменения и перезагрузив браузер, вы получите требуемый результат как при использовании регулярных выражений, так и при использовании стандартной строковой замены (рис. 9.4).

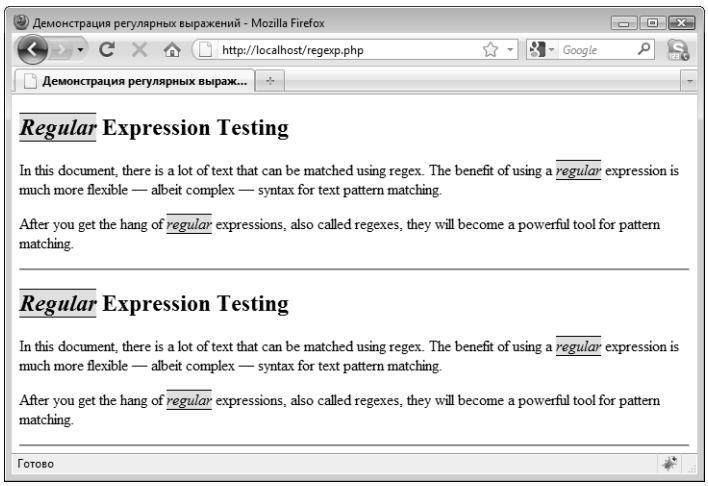



**Рис. 9.4.** Пример более сложной замены

**Примечание.** В оставшихся примерах этой главы используются только регулярные выражения.

## Поиск соответствий шаблону с помощью символьных классов

В некоторых случаях нужен не просто поиск определенных слов. Например, иногда необходима проверка того, что были использованы только символы из определенного диапазона (скажем, требуется удостовериться, что при указании телефонного номера были использованы только цифры или что в поле имени пользователя не встречаются никакие специальные символы).

Регулярные выражения позволяют определить символьный класс, представляющий собой набор символов, заключенный в квадратные скобки. Например, для поиска любого символа в диапазоне от `a` до `c` следует использовать выражение `[a-c]`.

Модифицируем файл `regexp.php` для выделения символов, принадлежащих диапазону `A-C`. Дополнительно поместим шаблон в переменную и выведем ее значение под данными примера; это позволит увидеть, какой шаблон использовался при загрузке сценария. Добавьте в файл код, выделенный ниже полужирным шрифтом.

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <title>Демонстрация регулярных выражений</title>
    <style type="text/css">
        em {
            background-color: #FF0;
            border-top: 1px solid #000;
            border-bottom: 1px solid #000;
        }
    </style>
</head>

<body>
<?php

/*
 * Строка для хранения текста, используемого в примерах
 * работы с регулярными выражениями
 */

$string = <<<TEST_DATA

<h2>Тестирование регулярных выражений</h2>
<p>
    In this document, there is a lot of text that can be matched
    using regex. The benefit of using a regular expression is much
    more flexible &mdash; albeit complex &mdash; syntax for text
    pattern matching.
</p>
<p>
```

```
    After you get the hang of regular expressions, also called
    regexes, they will become a powerful tool for pattern matching.
</p>
<hr />
TEST_DATA;

/*
 * Использовать регулярное выражение для визуального выделения
 * всех вхождений букв a-c
 */
$pattern = "/([a-c])/i";
echo preg_replace($pattern, "<em>$1</em>", $string);

/*
 * Вывести использованный шаблон
 */
echo "\n<p>Использованный шаблон: <strong>$pattern</strong></p>";

?>

</body>

</html>
```

После перезагрузки страницы символы, принадлежащие указанному диапазону, окажутся выделенными (рис. 9.5). Тот же результат будет получен, если использовать выражения `[abc]`, `[bac]` или любую другую перестановку этих символов, поскольку класс будет соответствовать любому из входящих в него символов. Кроме того, поскольку используется модификатор, отменяющий учет регистра букв (`i`), то для поиска соответствий шаблону, независимо от их регистра, в использовании выражения `[A-Ca-c]` нет никакой необходимости.

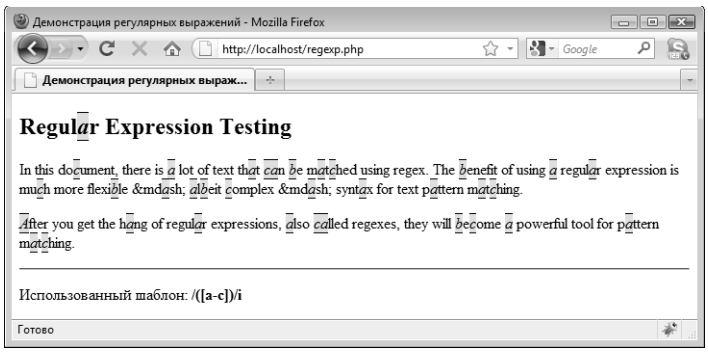



**Рис. 9.5.** Выделение всех символов из диапазона A–C

## *Поиск соответствий любым символам, кроме указанных*

Чтобы шаблону соответствовали любые символы, кроме тех, которые принадлежат указанному символьному классу, следует поместить перед классом символ вставки (`^`). Для выделения любых символов, кроме `A-C`, необходимо использовать шаблон `/([^a-c])/i` (рис. 9.6).

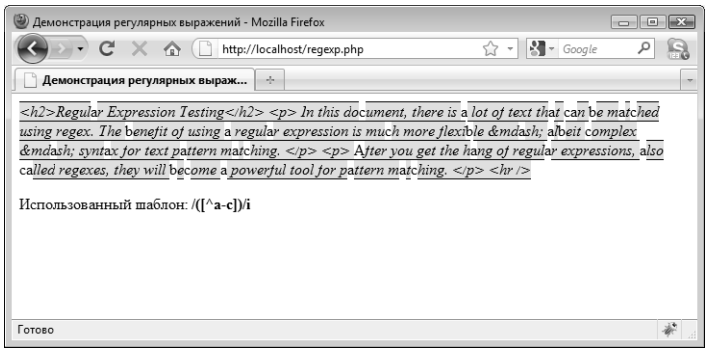



**Рис. 9.6.** Выделение всех символов, кроме A–C

**Примечание.** Важно подчеркнуть, что в предыдущих примерах символьный класс заключался в *круглые скобки*. Классы символов не имеют обратных ссылок, хранящихся вместе с ними, поэтому, чтобы сохранить возможность ссылаться на классы, следует использовать круглые скобки.

### Сокращенные обозначения символьных классов

Для некоторых символьных классов предусмотрены специальные сокращенные обозначения. Например, существуют специальные классы для символов, входящих в состав слова, а также для цифровых и пробельных символов.

- *Класс символов, образующих слова (алфавитно-цифровые символы)* (`\w`). Соответствует шаблонам наподобие `[A-Za-z0-9_]`.
- *Класс цифровых символов* (`\d`). Соответствует шаблонам наподобие `[0-9_]`.
- *Класс пробельных символов* (`\s`). Соответствует шаблонам наподобие `[ \t\r\n]`.

Использование этих трех специальных классов облегчает чтение регулярных выражений, особенно в случае сложных шаблонов.

Также можно исключить определенный тип символов, использовав в обозначении класса прописную букву.

- *Класс символов, не образующих слова.* Соответствует шаблонам наподобие `[^A-Za-z0-9_]`.
- *Класс нецифровых символов* (`\D`). Соответствует шаблонам наподобие `[^0-9_]`.
- *Класс непробельных символов* (`\S`). Соответствует шаблонам наподобие `[^ \t\r\n]`.

**Примечание.** Символы `\t`, `\r` и `n` — это специальные символы, представляющие символы табуляции и символы перехода на новую строку; пробел представляется обычным символом пробела ( ).

## Нахождение границ слов

Еще один специальный символ, о котором следует знать, — это символ границы слова (`\b`). Помещая его до и (или) после шаблона, вы гарантируете, что этот шаблон не будет содержаться *посреди* другого слова. Например, если нужно найти вхождения слова *stat*, но не слов *thermostat*, *statistic* или *ecstatic*, то необходимо использовать следующий шаблон: `/\bstat\b/`.

## Операторы повторения

Когда используется символьный класс, осуществляется поиск вхождений лишь одного символа из указанного набора, если только в шаблоне не указано другое количество символов. Регулярные выражения предоставляют в ваше распоряжение несколько способов для указания количества символов:

- оператору "звездочка" (*) соответствует произвольное количество вхождений предыдущего символа или полное его отсутствие;
- оператору "плюс" (+) соответствует одно или большее количество вхождений предыдущего символа;
- специальный оператор повторений (`{мин,макс}`) позволяет указать диапазон допустимого количества вхождений предыдущего символа.

Первый из перечисленных операторов полезен при поиске строк, которые могут содержать, а могут и не содержать определенную часть шаблона. Например, если необходимо найти все вхождения любой из строк *John* или *John Doe*, то можно использовать следующий шаблон, которому соответствуют обе строки: `/John( Doe)*/`.

Поиск одиночных или многократных вхождений символа пригодится тогда, когда следует убедиться в том, что был введен хотя бы один символ. Например, если вы хотите проверить, что пользователь ввел в элемент ввода формы хотя бы один символ и этот символ является допустимым алфавитно-цифровым символом, можете использовать следующий шаблон: `/\w+/`.

Наконец, оператор повторений оказывается особенно полезным при поиске вхождений численных значений, принадлежащих заданному интервалу. Например, для проверки того, что некоторое значение принадлежит интервалу от `0` до `99`, можно использовать следующий шаблон: `/\bd{1,2}\b/`.

Найдите в тестовом файле слова, состоящие в точности из четырех букв, с помощью следующего шаблона регулярного выражения: `/(\b\w{4}\b)/` (рис. 9.7).

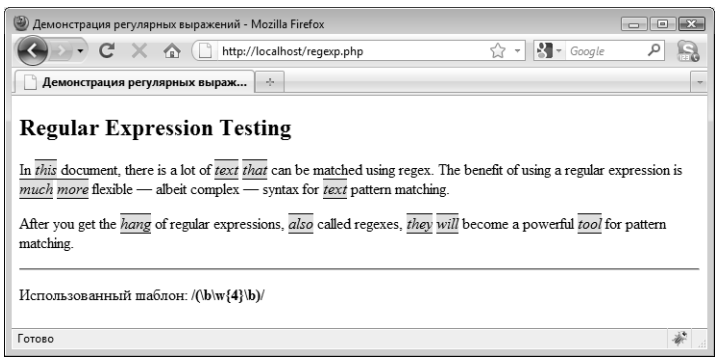



**Рис. 9.7.** Поиск слов, состоящих в точности из четырех букв

## Обнаружение начала и конца строки

Дополнительно можно также указать, что искомые вхождения шаблона должны располагаться в начале или в конце строки (или и там, и там). Если шаблон начинается с символа вставки (^), то соответствие будет найдено в том случае, если вхождение предыдущего символа располагается в начале строки. Если же шаблон заканчивается символом доллара ($), то соответствие будет найдено в том случае, если вхождение предыдущего символа располагается в конце строки.

Использование комбинации этих символов позволяет убедиться в том, что вся строка соответствует шаблону. Это особенно полезно при проверке ввода, поскольку позволяет удостовериться в том, что пользователь ввел только допустимые данные. Например, для проверки того, что введенное имя пользователя состоит только из букв A–Z, цифр 0–9 и символа подчеркивания, можно использовать следующий шаблон: `/^\w+$/`.

## Использование альтернативных шаблонов

В некоторых случаях желательно использовать либо один, либо другой шаблон. Такие шаблоны называются альтернативными и реализуются за счет использования символа канала, т.е. вертикальной черты (`|`). Это позволяет определить несколько возможностей при поиске соответствий шаблону. Например, чтобы найти в файле `regex.php` вхождения только трех-, шести- и семибуквенных слов, можно использовать следующий шаблон: `/\b(\w{3}|\w{6,7})\b/` (рис. 9.8).

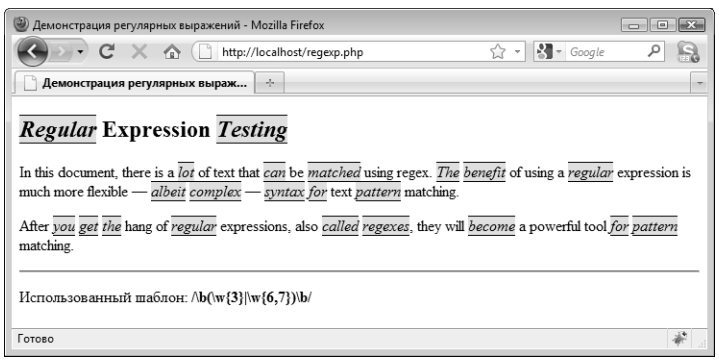



**Рис. 9.8.** Использование альтернативных шаблонов для поиска слов, состоящих из трех, шести или семи букв

## Использование необязательных элементов

Иногда требуется рассматривать некоторые элементы как необязательные. Например, для поиска форм единственного и множественного числа таких слов, как *expression*, необходимо сделать символ *s* необязательным.

В подобных случаях следует поместить после необязательного элемента вопросительный знак (`?`). Если необязательная часть шаблона включает в себя более одного символа, то она должна указываться в виде группы (об этом рассказано в следующем разделе).

А пока что выделите все вхождения слов *expression* и *expressions* с помощью следующего шаблона: `/(expressions?)/i` (рис. 9.9).

## Сводим все вместе

Теперь, когда вы уже имеете общее представление о регулярных выражениях, самое время попрактиковаться в применении накопленных знаний на примере составления шаблона регулярного выражения, который будет находить вхождения фраз *regular expression* или *regex*, в том числе и тех, в которых существительные употреблены во множественном числе.

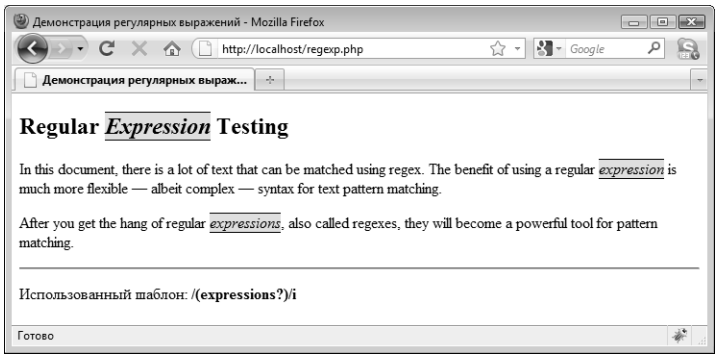



**Рис. 9.9.** Поиск соответствий шаблону с необязательным "s" в конце слова

Сначала организуем поиск слова *regex*: `/(regex)/` (рис. 9.10).

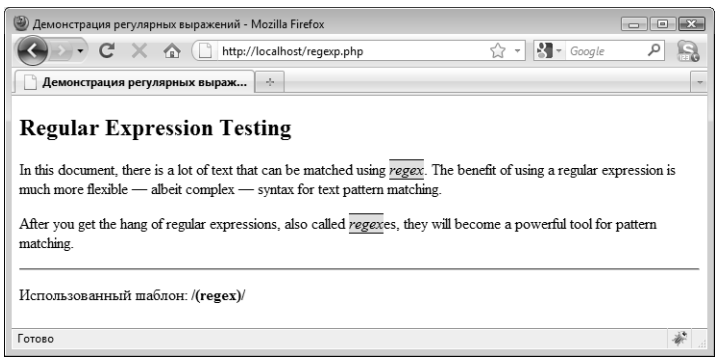



**Рис. 9.10.** Поиск вхождений слова "regex"

Затем дополнительно предусмотрим поиск существительных во множественном числе, вставив в конце шаблона необязательные символы `es`: `/(regex(es)?)/i` (рис. 9.11).

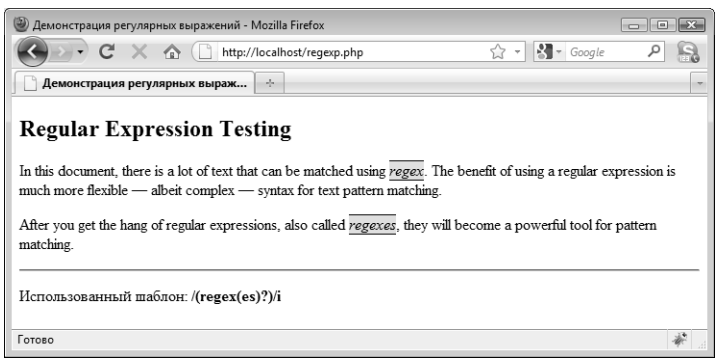



**Рис. 9.11.** Добавление необязательного соответствия для поиска существительных, употребленных во множественном числе

Далее дополним шаблон так, чтобы он соответствовал также слову *regular* с пробелом после него; сделаем это соответствие также необязательным: `/(reg(ular\s)?ex(es)?)/i` (рис. 9.12).

После этого расширим шаблон так, чтобы он содержал слово *expression* как альтернативу *es*: `/(reg(ular\s)?ex(pression|es)?)/i` (рис. 9.13).

Наконец, добавим необязательное `s` в конце поискового элемента для *expression*: `/(reg(ular\s)?ex(pressions?|es)?)/i` (рис. 9.14).

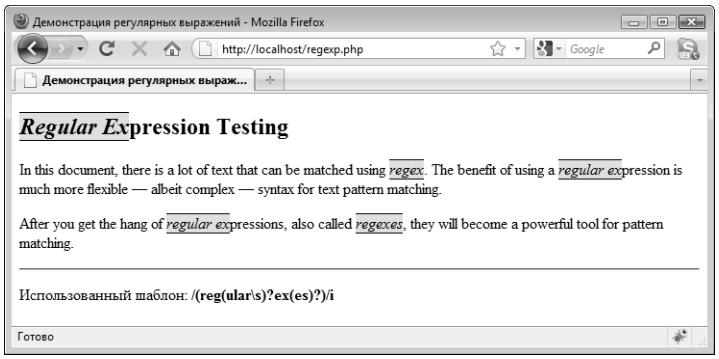



**Рис. 9.12.** Добавление проверки необязательного присутствия слова “regular”

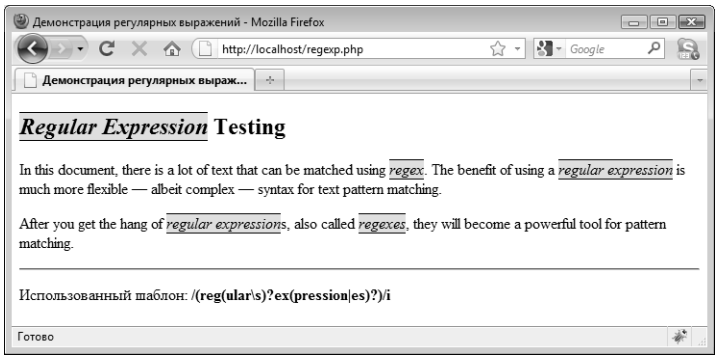



**Рис. 9.13.** Добавление альтернативного шаблона в поисковое выражение

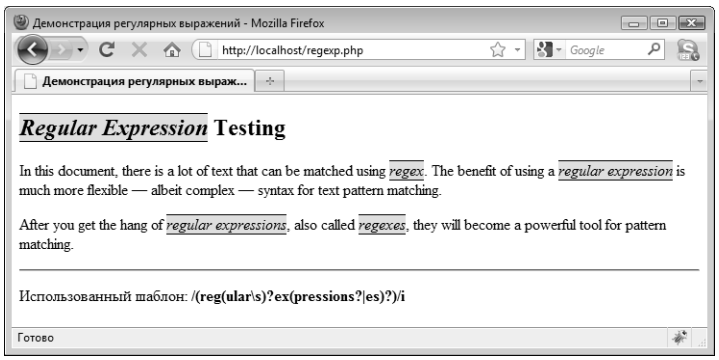



**Рис. 9.14.** Завершенное регулярное выражение

**Совет**. Хотя приведенные в этой главе примеры выходят за рамки наиболее часто используемых возможностей, они не охватывают всего, на что способны регулярные выражения. Фантастически полезный ресурс для изучения всех тонкостей регулярных выражений, а также некоторые инструментальные средства для их тестирования вы найдете на сайте `www.regular-expressions.info`.

# Проверка допустимости введенных значений даты и времени на стороне сервера

Теперь, когда вы получили все необходимые сведения, касающиеся работы с регулярными выражениями, можно приступить к проверке допустимости данных, вводимых пользователем. Нам понадобится верификация формата введенных дат, чтобы приложение не завершилось аварийно при попытке синтаксического разбора данных, которые оно не в состоянии понять.

Начнем с организации такой проверки на стороне сервера. В значительной степени это является подстраховкой, поскольку позднее будет добавлена проверка с использованием jQuery. Однако во всем, что имеет отношение к верификации пользовательского ввода, *никогда* нельзя целиком полагаться на JavaScript, поскольку пользователю ничего не стоит отключить поддержку JavaScript, а вместе с этим — и клиентские сценарии, осуществляющие верификацию данных.

## Определение шаблона регулярного выражения для проверки формата даты и времени

Первое, что необходимо сделать для организации проверки допустимости введенных дат, — это определить шаблон, по которому будет проверяться соответствие данных установленному формату. В нашем приложении для календаря используется формат дат `ГГГГ-ММ-ДД ЧЧ:ММ:СС`.

### Подготовка тестовых данных

Для тестирования шаблона нам потребуется модифицировать файл `regexp.php`, введя в него действительный формат и несколько недействительных. Начнем с создания шаблона для поиска соответствий любому количеству цифровых символов или их отсутствию. Для этого внесем в файл следующие изменения, выделенные полужирным шрифтом.

```
<!DOCTYPE html
 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <title>Демонстрация регулярных выражений</title>
    <style type="text/css">
        em {
            background-color: #FF0;
            border-top: 1px solid #000;
            border-bottom: 1px solid #000;
```

```
        }
    </style>
</head>

<body>
<?php

/*
 * Задать несколько строк тестовых дат для проверки работы
 * регулярных выражений
 */
$date[] = '2010-01-14 12:00:00';
$date[] = 'Saturday, May 14th at 7pm';
$date[] = '02/03/10 10:00pm';
$date[] = '2010-01-14 102:00:00';

/*
 * Шаблон для проверки допустимости дат
 */
$pattern = "/(\d*)/";

foreach ( $date as $d )
{
    echo "<p>", preg_replace($pattern, "<em>$1</em>", $d), "</p>";
}

/*
 * Вывести использованный шаблон
 */
echo "\n<p>Использованный шаблон: <strong>$pattern</strong></p>";

?>

</body>

</html>
```

Сохранив этот код, перезагрузите страницу `http://localhost/regex.php` в браузере, в котором должны отобразиться выделенные цифровые символы (рис. 9.15).

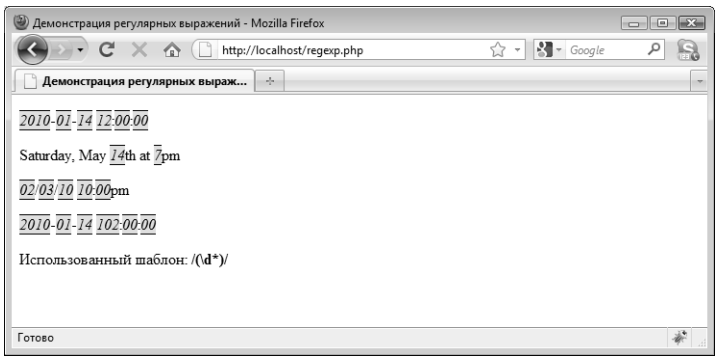



**Рис. 9.15.** Поиск соответствий цифровым символам

### Поиск соответствий формату даты и времени

Проверку совпадения с форматом даты начнем с элементов, содержащих ровно четыре цифры в начале строки, соответствующие году: `/^(\d{4})/` (рис. 9.16).

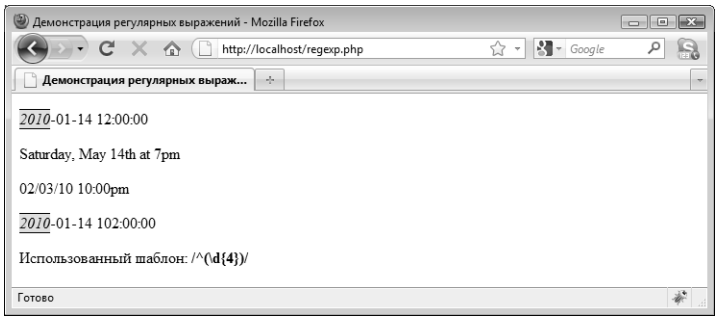



**Рис. 9.16.** Проверка правильности указания года в строке даты

После этого проверим правильность указания месяца, выполнив поиск дефиса и двух дополнительных цифр: `/^(\d{4}(-\d{2}))/` (рис. 9.17).


**Рис. 9.17.** Проверка правильности формата ввода месяца в строке даты

Заметьте, что подшаблоны для месяца и числа идентичны: дефис, за которым следуют две цифры. Это означает, что для проверки числа можно просто повторить подшаблон для месяца, применив оператор повторения после группы: `/^(\d{4}(-\d{2}){2})/` (рис. 9.18).


**Рис. 9.18.** Проверка правильности формата ввода числа в строке даты

Далее добавим подшаблон для поиска пробела и часа: `/^(\d{4}(-\d{2}){2} (\d{2}))/` (рис. 9.19).

**Примечание.** Убедитесь в том, что в шаблон включен символ пробела. Специальный класс `(\s)` для этого использовать нельзя, поскольку символы перехода на новую строку и символы табуляции в данном случае не соответствуют шаблону.

Для проверки минут добавим подшаблон, осуществляющий поиск двоеточия и следующих за ним двух цифр: `/^(\d{4}(-\d{2}){2} (\d{2})(:\d{2}))` (рис. 9.20).

Наконец, чтобы проверить формат ввода секунд, повторим подшаблон для проверки минут, а затем используем модификатор в виде знака доллара для поиска конца строки: `/^(\d{4}(-\d{2}){2} (\d{2})(:\d{2}){2})$/` (рис. 9.21).


**Рис. 9.19.** Проверка правильности формата ввода часа в строке даты

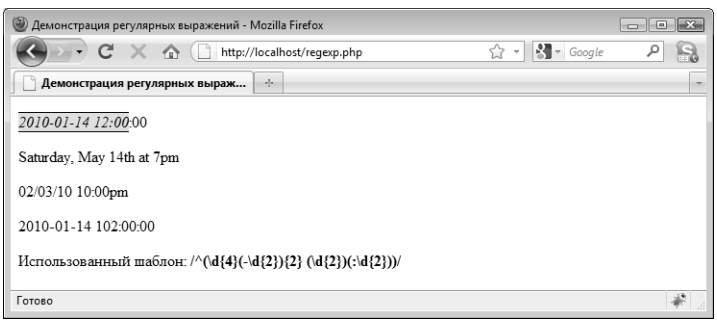



**Рис. 9.20.** Проверка правильности формата ввода минут в строке даты

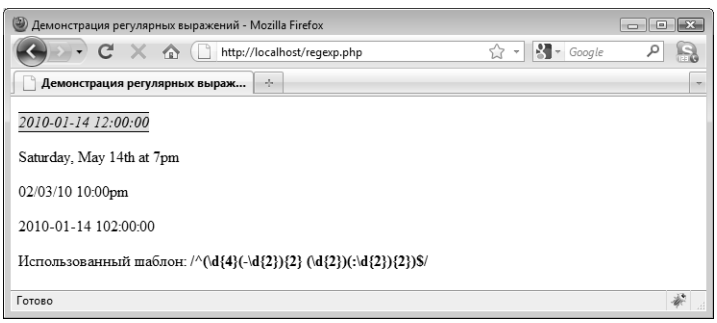



**Рис. 9.21.** Проверка правильности формата ввода секунд в строке даты

Вооруженные только что созданным шаблоном, можем приступать к проверке ввода дат в нашем приложении.

## Добавление метода проверки в класс `Calendar`

Для верификации строки даты добавим в класс `Calendar` новый закрытый метод `_validDate()`.

Этот метод будет принимать строку даты, подлежащую проверке, и сравнивать ее с шаблоном, используя для этого функцию `preg_match()`, которая возвращает количество совпадений, обнаруженных в данной строке. Поскольку шаблон будет соответствовать найденному образцу лишь тогда, когда с ним согласуется целиком вся строка, то в случае допустимой даты возвращаемым значением будет `1`, а в случае недопустимой — `0`.

Если дата допустима, наш метод возвратит `TRUE`, иначе — `FALSE`.

Добавьте этот метод в класс `Calendar`, вставив в файл `class.calendar.inc.php` следующий код, выделенный полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    public function buildCalendar() {...}

    public function displayEvent($id) {...}

    public function displayForm() {...}

    public function processForm() {...}

    public function confirmDelete($id) {...}

/**
 * Проверяет допустимость строки даты
 *
 * @param string $date: проверяемая строка даты
 * @return bool: успешное выполнение -- TRUE, иначе -- FALSE
 */
private function _validDate($date)
{
    /*
     * Определить шаблон регулярного выражения для проверки
     * формата даты
     */
```

```
    $pattern = '/^(\d{4}(-\d{2}){2} (\d{2})(:\d{2}){2})$/';

    /*
     * В случае соответствия возвратить TRUE, иначе -- FALSE
     */
    return preg_match($pattern, $date)==1 ? TRUE : FALSE;
}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

    private function _adminGeneralOptions() {...}

    private function _adminEntryOptions($id) {...}

}

?>
```

## Возврат ошибки в случае недопустимости даты и времени

Следующий шаг состоит в видоизменении метода `processForm()`, обеспечивающим вызов метода `_validDate()` для времени начала и завершения нового события. В случае неудачного завершения проверки метод будет возвращать сообщение об ошибке.

Для реализации указанной выше проверки введите в метод `processForm()` следующие изменения, выделенные полужирным шрифтом.

```
<?php

class Calendar extends DB_Connect
{

    private $_useDate;

    private $_m;

    private $_y;

    private $_daysInMonth;

    private $_startDay;

    public function __construct($dbo=NULL, $useDate=NULL) {...}

    public function buildCalendar() {...}

    public function displayEvent($id) {...}

    public function displayForm() {...}
```

```
    /**
     * Предназначен для проверки формы и сохранения или
     * редактирования события
     *
     * @return mixed: TRUE в случае успешного завершения или
     * сообщение об ошибке в случае сбоя
     */
    public function processForm()
    {
        /*
         * Выход, если параметр "action" задано неправильно
         */
        if ( $_POST['action']!='event_edit' )
        {
            return "Некорректная попытка вызова метода processForm";
        }

        /*
         * Извлечь данные из формы
         */
        $title = htmlentities($_POST['event_title'], ENT_QUOTES);
        $desc = htmlentities($_POST['event_description'], ENT_QUOTES);
        $start = htmlentities($_POST['event_start'], ENT_QUOTES);
        $end = htmlentities($_POST['event_end'], ENT_QUOTES);

/*
 * Если начальная или конечная даты не соответствуют допустимому
 * формату, завершить сценарий с возвратом сообщения об ошибке
 */
if ( !$this->_validDate($start)
        || !$this->_validDate($end) )
{
    return "Недействительный формат даты! Используйте следующий
            формат: ГГГГ-ММ-ДД ЧЧ:ММ:СС";
}

        /*
         * Если ID не был передан, создать новое событие
         */
        if ( empty($_POST['event_id']) )
        {
            $sql = "INSERT INTO `events`
                        (`event_title`, `event_desc`, `event_start`,
                            `event_end`)
                    VALUES
                        (:title, :description, :start, :end)";
        }

        /*
         * Обновить событие, если оно редактировалось
         */
        else
        {
            /*
```

```
             * Привести ID события к целочисленному типу в интересах безопасности
             */
            $id = (int) $_POST['event_id'];
            $sql = "UPDATE `events`
                    SET
                        `event_title`=:title,
                        `event_desc`=:description,
                        `event_start`=:start,
                        `event_end`=:end
                    WHERE `event_id`=$id";
        }

        /*
         * После привязки данных выполнить запрос создания или
         * редактирования события
         */
        try
        {
            $stmt = $this->db->prepare($sql);
            $stmt->bindParam(":title", $title, PDO::PARAM_STR);
            $stmt->bindParam(":description", $desc, PDO::PARAM_STR);
            $stmt->bindParam(":start", $start, PDO::PARAM_STR);
            $stmt->bindParam(":end", $end, PDO::PARAM_STR);
            $stmt->execute();
            $stmt->closeCursor();

            /*
             * Возвратить ID события
             */
            return $this->db->lastInsertId();
        }
        catch ( Exception $e )
        {
            return $e->getMessage();
        }
    }

    public function processForm() {...}

    public function confirmDelete($id) {...}

    private function _validDate($date) {...}

    private function _loadEventData($id=NULL) {...}

    private function _createEventObj() {...}

    private function _loadEventById($id) {...}

    private function _adminGeneralOptions() {...}

    private function _adminEntryOptions($id) {...}

}

?>
```

Протестируйте работу процедуры верификации, используя заведомо неправильный формат даты и времени в форме на странице `http://localhost/admin.php` (рис. 9.22).

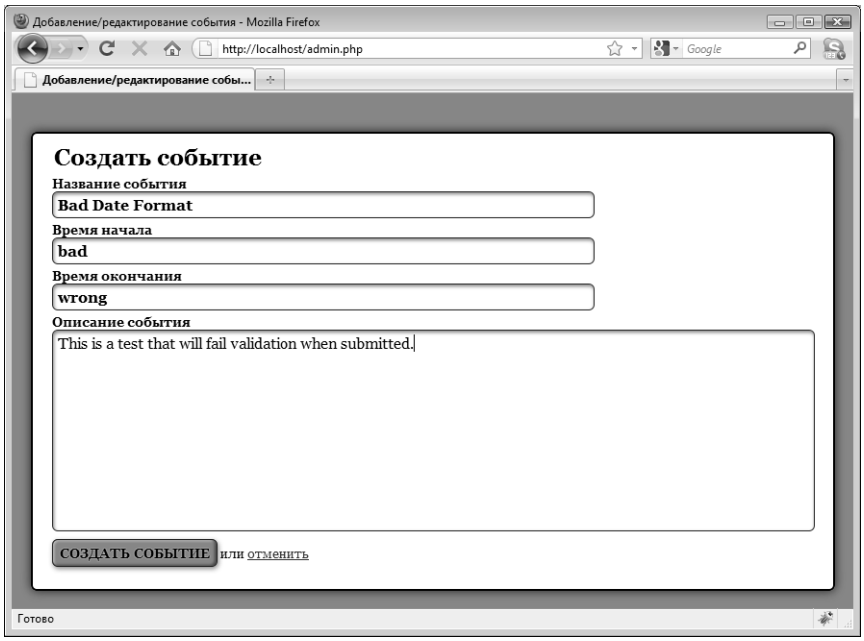



**Рис. 9.22.** Значения даты и времени, заданные в недопустимом формате, не выдержат проверки

**Примечание.** Причина, по которой мы использовали страницу `http://localhost/admin.php`, заключается в том, что проверка допустимости данных на серверной стороне будет запускаться, лишь если пользователь запретил использование сценариев JavaScript на стороне клиента. В этом случае модальные окна не будут функционировать, и пользователь будет перенаправлен на эту страницу. В ситуациях, когда работа сценариев JavaScript разрешена, серверная сторона выполняет проверку для подстраховки, используя ее в качестве дополнительной меры защиты приложения от действий злоумышленников.

После отправки формы приложение просто выведет сообщение об ошибке и прекратит работу (рис. 9.23). Календарь ориентирован на пользователей, работающих с включенными сценариями JavaScript; вы используете этот подход для того, чтобы приложение не выводило сообщений об ошибках.

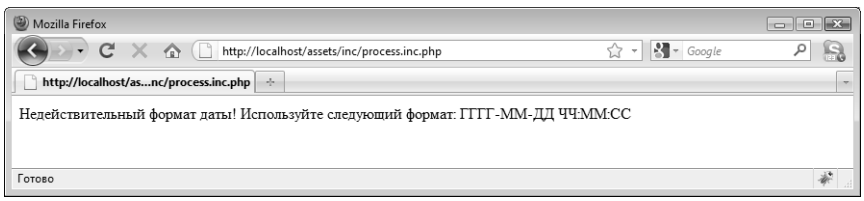



**Рис. 9.23.** Отображение сообщения об ошибке в случае предоставления недопустимых значений даты и времени

# Проверка допустимости задания даты и времени на стороне клиента

На компьютерах большинства пользователей выполнение сценариев JavaScript будет разрешено. Пользователю будет намного удобнее работать, если он сможет рассчитывать на немедленную ответную реакцию формы, в связи с чем мы добавим в приложение новую функциональность jQuery, обеспечивающую проверку допустимости введенных строк даты и времени на стороне клиента.

## Создание нового файла сценария JavaScript для проверки допустимости значений даты и времени

Поскольку мы собираемся продолжить работу с этим сценарием в следующей главе, предусмотрим для него отдельный файл `valid-date.js`, который поместим в папку `js`. Этот файл будет содержать функцию, являющуюся функциональным эквивалентом метода `_validDate()` класса `Calendar`.

Сценарий будет принимать дату, подлежащую проверке, сверять ее с шаблоном регулярного выражения, который мы ранее создали, используя функцию `match()`, а затем возвращать значение `true` в случае соответствия даты шаблону или `false`, если функция `match()` возвратит `null`.

Для создания этой функции введите в файл `valid_date.js` следующий код.

```
// Проверяет допустимость строки даты и времени
// (ГГГГ-ММ-ДД ЧЧ:ММ:СС)
function validDate(date)
{
    // Определить шаблон регулярного выражения для проверки формата
    var pattern = /^(\d{4}(-\d{2}){2} (\d{2})(:\d{2}){2})$/;

    // Возвратить true, если формат даты совпадает, или
    // false в противном случае
    return date.match(pattern)!=null;
}
```

**Примечание.** Шаблон регулярного выражения не заключается в кавычки. Если использовать кавычки, то шаблон сохранится как строка и будет соответствующим образом интерпретироваться, т.е. сценарий будет выполнять поиск точного совпадения строк, а не интерпретировать шаблон, как это следовало бы делать.

## Включение нового файла в завершающую часть страницы

Чтобы функция `validDate()` была доступна для вызова, новый JavaScript-файл должен включаться в сценарий до файла `init.js`. Для этого откройте файл `footer.inc` и вставьте в него следующий код, выделенный полужирным шрифтом.

```
<script type="text/javascript"
    src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
    google.load("jquery", "1");
</script>
<script type="text/javascript"
src="assets/js/valid-date.js"></script>
```

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

</html>
```

## Предотвращение отправки формы в случае отрицательного результата проверки

Теперь, когда в нашем распоряжении имеется функция `validDate()`, доступная в файле `init.js`, добавим проверку формата введенных даты и времени перед отправкой формы. Сохраним даты начала и завершения события в переменных (`start` и `end` соответственно), а затем проверим их с помощью функции `validDate()`.

Далее необходимо модифицировать обработчик события `click` кнопки отправки формы, предназначенной для создания и редактирования событий, а затем вывести сообщение об ошибке, если введенные значения даты и времени являются недопустимыми. В этом случае потребуется также запретить отправку формы, чтобы пользователю не пришлось заново вводить информацию в остальные поля формы.

Для реализации этого введите в файл `init.js` следующий код.

```
// Удостовериться в готовности документа, прежде чем выполнять сценарии
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",
    fx = {...}

$("li a").live("click", function(event){...});

$(".admin-options form,.admin")
.live("click", function(event){...});

// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Сериализовать данные формы для использования
        // с функцией $.ajax()
        var formData = $(this).parents("form").serialize(),

        // Сохранить значение кнопки "submit"
        submitVal = $(this).val(),

        // Сохранить введенную строку даты начала события
        start = $(this).siblings("[name=event_start]").val(),

        // Сохранить введенную строку даты завершения события
        end = $(this).siblings("[name=event_end]").val();

        // Определить, подлежит ли событие исключению из календаря
        remove = false;
```

```
// Если это форма для удаления, присоединить действие
if ( $(this).attr("name")=="confirm_delete" )
{

    // Добавить необходимую информацию в строку запроса
    formData += "&action=confirm_delete"
        + "&confirm_delete="+submitVal;

    // Если событие действительно удаляется, установить
    // флаг для исключения его из разметки
    if ( submitVal=="Yes, Delete It" )
    {
        remove = true;
    }
}

// Если событие создается/редактируется,
// проверить действительность дат
if ( $(this).siblings("[name=action]").val()=="event_edit" )
{
    if ( !validDate(start) || !validDate(end) )
    {
        alert("Требуется действительная дата!
               (ГГГГ-ММ-ДД ЧЧ:ММ:СС)");
        return false;
    }
}

// Отправить данные обрабатывающему файлу
$.ajax({
        type: "POST",
        url: processFile,
        data: formData,
        success: function(data) {
            // Если это событие удалено, исключить
            // его из разметки
            if ( remove===true )
            {
                fx.removeevent();
            }
            // Обеспечить плавное исчезновение
            // модального окна
            fx.boxout();

            // Если это новое событие, добавить
            // его в календарь
            if ( $("[name=event_id]").val().length==0
                && remove===false )
            {
                fx.addevent(data, formData);
            }
        },
        error: function(msg) {
            alert(msg);
```

```
                }
            });
    });

$(".edit-form a:contains(отменить)")
    .live("click", function(event){...});

});
```

Сохраните изменения, загрузите страницу `http://localhost/` и создайте новое событие с недопустимыми значениями параметров, используя форму модального окна (рис. 9.24).

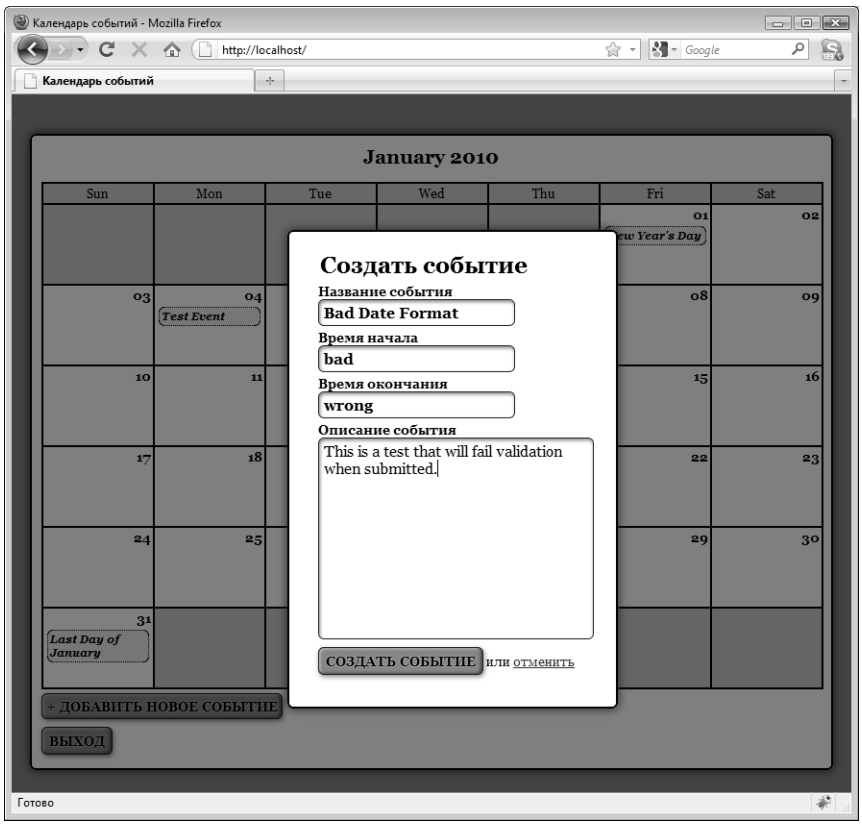



**Рис. 9.24.** Исходные данные, которые заведомо не пройдут проверку

Если теперь щелкнуть на кнопке отправки формы, то проверка завершится неудачно и приложение отобразит окно с предупреждающим сообщением относительно неверного формата даты (рис. 9.25).

После щелчка на кнопке OK окна предупреждения пользователь сможет отредактировать данные, не вводя повторно информацию в остальные поля.

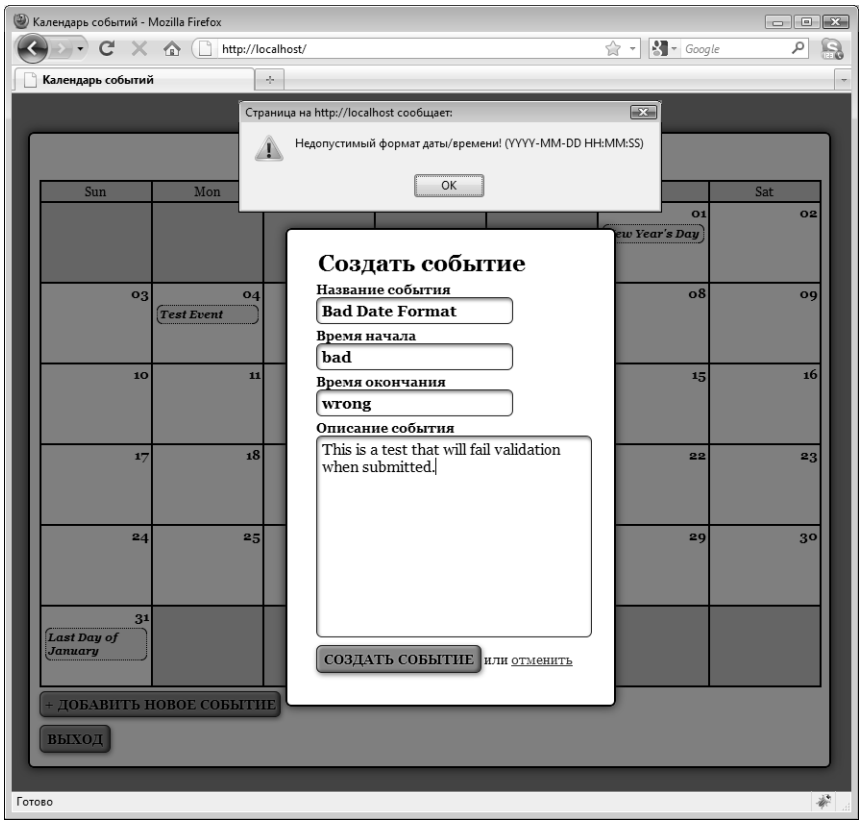



**Рис. 9.25.** Вывод сообщения об ошибке после неудачного завершения верификации даты

# Резюме

В этой главе вы усердно поработали с регулярными выражениями, используя их для проверки допустимости формы. Усвоенные вами идеи с успехом могут применяться для проверки допустимости любых типов данных и станут серьезным подспорьем в тех случаях, когда необходимо быть уверенным в том, что приложение сможет работать с предоставленными в форме данными.

В следующей главе вы научитесь расширять объект jQuery, причем сможете это делать как путем непосредственного расширения ядра jQuery, так и путем разработки подключаемых модулей для jQuery.

Глава 10

# Расширение jQuery

Простота синтаксиса jQuery побудила разработчиков к написанию сценариев для создания всевозможных эффектов и решения целого ряда других задач. Чтобы такие сценарии можно было конфигурировать и многократно использовать, их создают в виде дополнений, называемых *подключаемыми модулями* (*плагинами*), или сценариев, которые расширяют функциональность jQuery за счет добавления новых методов в библиотеку. В этой главе вы узнаете о том, как добавлять собственные подключаемые модули в jQuery.

## Добавление функций в jQuery

В некоторых случаях может оказаться желательным добавить какую-либо функцию непосредственно в объект jQuery, тем самым делая возможными вызовы функций наподобие следующего:

```
$.yourFunction();
```

Добавление функций в jQuery может способствовать лучшей организации сценариев и служить гарантией того, что вызовы любой такой функции всегда будут подчиняться одному и тому же формату. Однако, и это надо особо подчеркнуть, функцию, добавленную в jQuery, *нельзя* включать в цепочки вызовов с набором выбранных DOM-элементов. О том, какой метод следует для этого использовать, также рассказывается в этой главе.

### Добавление функции проверки даты и времени в jQuery

В качестве первого примера расширения jQuery рассмотрим добавление в объект jQuery функции проверки формата даты и времени (далее для краткости — *даты*), которая была создана в предыдущей главе и сохранена в файле `valid-date.js`.

#### *Применение определенных пользователем псевдонимов в дополнениях jQuery*

Приступая к разработке дополнений, стоит подумать об использовании в них собственных псевдонимов для объекта jQuery. И хотя эта мера не является строго обязательной, она *весьма* желательна, поскольку позволяет избежать возможных

неприятностей в тех случаях, когда контроль над использованием символа $ был передан другим библиотекам путем вызова метода jQuery.noConflict(). Более того, реализация этой возможности настолько проста, что не воспользоваться ею было бы просто неразумно.

Суть подхода сводится к тому, что при создании нового дополнения вы помещаете его код в тело функции, которая выполняется сразу же при загрузке сценария. Соответствующий сценарий должен с самого начала строиться по следующему образцу.

```
(function(){
    // здесь будет помещен код дополнения...
})();
```

Второй набор круглых скобок заставляет предыдущий код немедленно выполниться в виде функции, и именно здесь на первый план выходят псевдонимы. Если второму набору круглых скобок передать объект jQuery, а внутренней функции — его псевдоним $, то код будет правильно работать с этим псевдонимом даже в том случае, если он был возвращен в глобальное пространство имен в результате вызова метода jQuery.noConflict().

```
(function($){
    // здесь будет помещен код дополнения...
})(jQuery);
```

Вместо $ можно использовать любое другое допустимое имя переменной JavaScript, и сценарий по-прежнему будет нормально работать с этим методом.

```
(function(custom){
    // Добавить цвет фона в любой элемент абзаца,
    // используя пользовательский псевдоним
    custom("p").css("background-color","yellow");
})(jQuery);
```

### Присоединение функции к объекту jQuery

Чтобы присоединить проверочную функцию к jQuery, введите в файл valid-date.js следующий код.

```
(function($){

    // Расширяет объект jQuery для проверки допустимости строк даты
    $.validDate = function()
    {
        // здесь будет помещен код
    };

})(jQuery);
```

Использование такого формата дает возможность вызывать функцию следующим образом:

```
$.validDate();
```

### Добавление конфигурационных параметров

Как и в случае исходной функции validDate(), в новую функцию будет передаваться строка даты. Однако чтобы сделать эту функцию настраиваемой, ей можно будет (при необходимости) передать объект, содержащий конфигурационные пара-

метры, которые позволяют модифицировать шаблон регулярного выражения, используемого для проверки соответствия строки даты нужному формату.

```
(function($){

    // Расширяет объект jQuery для проверки допустимости строк даты
    $.validDate = function(date, options)
    {
        // здесь будет помещен код
    };

})(jQuery);
```

Объект `options` будет иметь только одно свойство: шаблон, используемый для проверки. Поскольку целесообразно сделать этот параметр необязательным, определим в функции значение по умолчанию для шаблона, вставив следующий код, выделенный полужирным шрифтом.

```
(function($){

    // Расширяет объект jQuery для проверки допустимости строк даты
    $.validDate = function(date, options)
    {
        // Установить значения по умолчанию для метода
        var defaults = {
            "pattern" : /^\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}$/
        };
    };

})(jQuery);
```

### Добавление параметров по умолчанию, предоставляемых пользователем

Можно расширить объект `default` с помощью функции `$.extend()`, которая создает новый объект, комбинируя параметры, заданные по умолчанию, с параметрами, предоставляемыми пользователем. Если имеются три параметра, а пользователь передает объект, в котором определены только два из них, то функция `$.extend()` заменит лишь два свойства, а именно те, которые переопределены пользователем.

Вставьте в объект `default` следующий код, выделенный полужирным шрифтом.

```
(function($){

    // Расширяет объект jQuery для проверки допустимости строк даты
    $.validDate = function(date, options)
    {
        // Установить значения по умолчанию для метода
        var defaults = {
            "pattern" : /^\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}$/
        },

        // Дополняет параметры по умолчанию пользовательскими
        // параметрами
        opts = $.extend(defaults, options);
    };

})(jQuery);
```

### Выполнение проверки и возврат значения

Эта процедура почти идентична той, которая использовалась в исходной версии верифицирующей функции, за исключением того, что в данном случае доступ к шаблону осуществляется посредством объекта `opts`.

```
(function($){

    // Расширяет объект jQuery для проверки допустимости строк даты
    $.validDate = function(date, options)
    {
        // Установить значения по умолчанию для метода
        var defaults = {
            "pattern" : /^\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}$/
        },

        // Дополнить параметры по умолчанию пользовательскими
        // параметрами
        opts = $.extend(defaults, options);

        // Если найдено совпадение, возвратить true, иначе -- false
        return date.match(opts.pattern)!=null;

    };

})(jQuery);
```

### Соблюдение правил именования файлов дополнений jQuery

Чтобы созданное дополнение удовлетворяло всем формальным требованиям, оно должно соответствовать принятым в jQuery правилам именования файлов дополнений. Допустимым является следующий формат: `jquery.[имя_дополнения].js`. Чтобы соблюсти его, изменим имя файла `valid_date.js` на `jquery.validDate.js`.

## Модификация сценария, выполняющего включение файлов

Поскольку имя файла было изменено, необходимо соответствующим образом обновить содержимое файла `footer.inc.php`, который включает файл дополнения в приложение. Внесите следующие изменения, выделенные полужирным шрифтом.

```
    <script type="text/javascript"
        src="http://www.google.com/jsapi"></script>
    <script type="text/javascript">
        google.load("jquery", "1");
    </script>
    <script type="text/javascript"
          src="assets/js/jquery.validDate.js"></script>
    <script type="text/javascript"
          src="assets/js/init.js"></script>
</body>

</html>
```

## Модификация сценария инициализации

Наконец, настройте файл `init.js` для вызова новой функции jQuery, которая была только что создана, и внесите в него следующие изменения, выделенные полужирным шрифтом.

```
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",

// Функции для манипулирования модальным окном
    fx = {...}

$("li>a").live("click", function(event){...}

$(".admin-options form,.admin").live("click", function(event){...}

// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]").live("click", function(event){

        // Предотвратить выполнение действия по умолчанию для формы
        event.preventDefault();

        // Сериализовать данные формы для использования
        // с функцией $.ajax()
        var formData = $(this).parents("form").serialize(),

        // Сохранить значение кнопки "submit"
        submitVal = $(this).val(),
        // Сохранить введенную строку даты начала события
        start = $(this).siblings("[name=event_start]").val(),

        // Сохранить введенную строку даты завершения события
        end = $(this).siblings("[name=event_end]").val();

        // Определить, подлежит ли событие исключению из календаря
        remove = false;

        // Если это форма для удаления, присоединить действие
        if ( $(this).attr("name")=="confirm_delete" )
        {

            // Добавить необходимую информацию в строку запроса
            formData += "&action=confirm_delete"
                + "&confirm_delete="+submitVal;

            // Если событие действительно удаляется, установить
            // флаг для исключения его из разметки
            if ( submitVal=="Yes, Delete It" )
            {
                remove = true;
            }
        }
        // Если событие создается/редактируется,
```

```
        // проверить действительность дат
        if ( $(this).siblings("[name=action]").val()=="event_edit" )
        {
            if ( !validDate(start) || !validDate(end) )
            {
                alert("Требуется действительная дата! (ГГГГ-ММ-ДД
                       ЧЧ:ММ:СС)");
                return false;
            }
        }

        // Отправить данные обрабатывающему файлу
        $.ajax({
                type: "POST",
                url: processFile,
                data: formData,
                success: function(data) {
                    // Если это событие удалено, исключить
                    // его из разметки
                    if ( remove===true )
                    {
                        fx.removeevent();
                    }
                    // Обеспечить плавное исчезновение
                    // модального окна
                    fx.boxout();

                    // Если это новое событие, добавить
                    // его в календарь
                    if ( $("[name=event_id]").val().length==0
                        && remove===false )
                    {
                        fx.addevent(data, formData);
                    }
                },
                error: function(msg) {
                    alert(msg);
                }
            });
    });

$(".edit-form a:contains(отменить)")
    .live("click", function(event){...};

});
```

Сохранив этот код, перезагрузите страницу `http//localhost/` и попытайтесь отправить форму с новым событием, для которого заданы недопустимые значения даты и времени. Результат совпадет с тем, который был получен с использованием исходной версии функции `validDate()`.

# Добавление методов в jQuery

Чтобы добавить в объект jQuery метод, пригодный для включения в цепочки вызовов, его необходимо присоединить к объекту `fn` jQuery. После этого метод можно будет вызывать из набора выбранных с помощью селектора элементов.

```
$(".class").yourPlugin();
```

**Примечание.** В действительности объект `fn` jQuery есть не что иное, как псевдоним другого объекта jQuery — `prototype`. Модификация прототипа любого объекта повлияет на все создаваемые впоследствии экземпляры этого объекта, а не только на текущий экземпляр. Для получения более подробной информации по этому вопросу ознакомьтесь с кратким описанием свойств объекта `prototype` в JavaScript, находящимся по следующему адресу: `http://www.javascriptkit.com/javatutors/proto.shtml`.

## Создание собственного подключаемого модуля

В основе создаваемого в этом разделе дополнения лежит простой метод, который увеличивает размер шрифта в названии события при наведении на него указателя и восстанавливает его первоначальный размер при смещении указателя сторону.

Это дополнение, которое мы назовем `dateZoom`, будет предоставлять пользователю возможность задавать размер увеличенного шрифта, а также скорость и уравнение изинга, используемые для создания эффекта анимации.

### Создание файла дополнения

Первое, что следует сделать, приступая к созданию дополнения, — это присвоить ему имя. Создайте новый файл в папке `js`, назовите его `jquery.dateZoom.js` и поместите в него следующую функцию.

```
(function($){
    // здесь будет помещен код дополнения
})(jQuery);
```

Внутри этой функции присоедините новый метод к объекту `fn` с помощью следующего кода, выделенного полужирным шрифтом.

```
(function($){

    // Дополнение, которое увеличивает размер текста элемента при
    // наведении на него указателя, а затем возвращает его к исходному
    // размеру при смещении указателя в сторону
    $.fn.dateZoom = function(options)
    {
        // здесь будет помещен код
    };

})(jQuery);
```

### Обеспечение общедоступности параметров по умолчанию

В дополнении `validDate` значения параметров функции, используемые по умолчанию, хранятся в закрытом объекте. Иногда это может оказаться нежелательным, если ожидается, что впоследствии может возникнуть необходимость в изменении этих значений, поскольку в таком случае к ним должен быть обеспечен доступ.

Чтобы сделать параметры, используемые по умолчанию, общедоступными, можно сохранить их в пространстве имен `dateZoom`. Создайте для дополнения `dateZoom` общедоступный объект `defaults`, содержащий следующие четыре свойства.

- **`fontsize`.** Степень увеличения размера шрифта. По умолчанию установите ее равной 110%.
- **`easing`.** Функция изинга, определяющая вид анимации. По умолчанию установите для нее значение `swing`.
- **`duration`.** Длительность анимации в миллисекундах. По умолчанию установите ее равной 600.
- **`callback`.** Функция, которая запускается по завершении анимации. По умолчанию установите для этого параметра значение `null`.

Добавьте параметры по умолчанию в дополнение `dateZoom`, введя следующий код, выделенный полужирным шрифтом.

```
(function($){

    // Дополнение, которое увеличивает размер текста элемента
    // при наведении на него указателя, а затем возвращает его
    // к исходному размеру при смещении указателя в сторону
    $.fn.dateZoom = function(options)
    {
        // здесь будет помещен код
    };

    // Определить значения по умолчанию для дополнения
    $.fn.dateZoom.defaults = {
            "fontsize" : "110%",
            "easing" : "swing",
            "duration" : "600",
            "callback" : null
        };

})(jQuery);
```

Пользователь может изменить значения по умолчанию для всех вызовов дополнения `dateZoom`, используя команды следующего типа:

```
$.fn.dateZoom.defaults.fontsize = "120%";
```

Чтобы переопределить значения параметров, установленные по умолчанию, пользователь может передать объект, содержащий новые значения для одного или нескольких параметров, как это было сделано в дополнении `validDate`. Для создания нового объекта, содержащего новые значения параметров по умолчанию для текущего вызова дополнения, можно использовать метод `$.extend()`.

Описанная функциональность вносится в дополнение `dateZoom` путем добавления следующего кода, выделенного полужирным шрифтом.

```
(function($){

    // Дополнение, которое увеличивает размер текста элемента
    // при наведении на него указателя, а затем возвращает его
    // к исходному размеру при смещении указателя в сторону
    $.fn.dateZoom = function(options)
    {
```

```
        // Переопределить лишь те значения, которые были явно
        // переданы пользователем в параметрах
        var opts = $.extend($.fn.dateZoom.defaults, options);

        // здесь будет помещен дополнительный код
    };

    // Определить значения по умолчанию для дополнения
    $.fn.dateZoom.defaults = {
            "fontsize" : "110%",
            "easing" : "swing",
            "duration" : "600",
            "callback" : null
        };

})(jQuery);
```

### Поддержка возможности включения метода в цепочки

Чтобы метод дополнения можно было включать в цепочки вызовов, он должен возвращать модифицированный объект `jQuery`. К счастью, реализовать это в jQuery не составляет труда: все, что для этого надо сделать, — вызвать метод `.each()` из объекта `this` для итеративного перебора всех элементов текущего найденного набора, а затем возвратить объект `this`.

В случае дополнения `dateZoom` можете сделать свой метод пригодным для включения в цепочки, внеся в код следующие изменения, выделенные полужирным шрифтом.

```
(function($){

    // Дополнение, которое увеличивает размер текста элемента
    // при наведении на него указателя, а затем возвращает его
    // к исходному размеру при смещении указателя в сторону
    $.fn.dateZoom = function(options)
    {
        // Переопределить лишь те значения, которые были явно
        // переданы пользователем в параметрах
        var opts = $.extend($.fn.dateZoom.defaults, options);

        // Организовать цикл по всем выбранным элементам
        // и вернуть модифицированный объект jQuery для
        // поддержки возможности вызова в цепочках
        return this.each(function(){
            // здесь будет помещен дополнительный код
        });
    };

    // Определить значения по умолчанию для дополнения
    $.fn.dateZoom.defaults = {
            "fontsize" : "110%",
            "easing" : "swing",
            "duration" : "600",
            "callback" : null
        };

})(jQuery);
```

### Создание общедоступного вспомогательного метода

Чтобы обеспечить удобочитаемость и структурированность кода дополнения, поместим анимацию во вспомогательный метод, который назовем `zoom`.

Как и объект `defaults`, этот метод будет общедоступным в пространстве имен `dateZoom`. Общедоступность метода означает, что у пользователя появляется потенциальная возможность переопределить его до вызова дополнения и даже вызвать извне дополнения, если в этом возникнет потребность.

Создайте метод `zoom`, вставив в дополнение `dateZoom` следующий код, выделенный полужирным шрифтом.

```
(function($){

    // Дополнение, которое увеличивает размер текста элемента
    // при наведении на него указателя, а затем возвращает его
    // к исходному размеру при смещении указателя в сторону
    $.fn.dateZoom = function(options)
    {
        // Переопределить лишь те значения, которые были явно
        // переданы пользователем в параметрах
        var opts = $.extend($.fn.dateZoom.defaults, options);

        // Организовать цикл по всем выбранным элементам
        // и вернуть модифицированный объект jQuery для
        // поддержки возможности вызова в цепочках
        return this.each(function(){
            // здесь будет помещен дополнительный код
        });
    };

    // Определить значения по умолчанию для дополнения
    $.fn.dateZoom.defaults = {
            "fontsize" : "110%",
            "easing" : "swing",
            "duration" : "600",
            "callback" : null
        };

    // Определить вспомогательную функцию, к которой пользователь
    // при необходимости сможет получить доступ извне дополнения
    $.fn.dateZoom.zoom = function(element, size, opts)
    {
        // увеличить размер элементов
    };

})(jQuery);
```

Аргументами этого метода служат элемент, подлежащий анимации, степень увеличения размера элемента и объект, содержащий значения параметров.

---

**Примечание.** Параметр, определяющий степень увеличения размера, отделен от остальных параметров по той причине, что для возврата элемента в первоначальное состояние будет использоваться его исходный размер шрифта, а этот параметр недоступен в объекте `options`.

---

Внутри этого метода для анимации объекта и предотвращения накопления объектов в очереди анимации будут использоваться методы `.animate()`, `.dequeue()`

и `clearQueue()`. Чтобы реализовать это, вставьте следующий код, выделенный полужирным шрифтом.

```
(function($){

    // Дополнение, которое увеличивает размер текста элемента
    // при наведении на него указателя, а затем возвращает его
    // к исходному размеру при смещении указателя в сторону
    $.fn.dateZoom = function(options)
    {
        // Переопределить лишь те значения, которые были явно
        // переданы пользователем в параметрах
        var opts = $.extend($.fn.dateZoom.defaults, options);

        // Организовать цикл по всем выбранным элементам
        // и вернуть модифицированный объект jQuery для
        // поддержки возможности вызова в цепочках
        return this.each(function(){
            // здесь будет помещен дополнительный код
        });
    };

    // Определить значения по умолчанию для дополнения
    $.fn.dateZoom.defaults = {
            "fontsize" : "110%",
            "easing" : "swing",
            "duration" : "600",
            "callback" : null
        };

    // Определить вспомогательную функцию, к которой пользователь
    // при необходимости сможет получить доступ извне дополнения
    $.fn.dateZoom.zoom = function(element, size, opts)
    {

        $(element).animate({
                    "font-size" : size
                },{
                    "duration" : opts.duration,
                    "easing" : opts.easing,
                    "complete" : opts.callback
                })
            .dequeue() // Предотвратить скачки анимации
            .clearQueue(); // Обеспечить выполнение только
                           // одной анимации
    };

})(jQuery);
```

**Примечание.** Метод `.dequeue()` исключает текущую анимацию из очереди во избежание ее перемещения в конец очереди, когда та очищается с помощью метода `.clearQueue()`. Накопление объектов в очереди анимации нежелательно тем, что это может приводить к скачкообразному выполнению анимации или ее быстрому многократному повторению, что явно нежелательно.

### Поочередное видоизменение каждого из выбранных элементов

Поскольку одним из параметров, передаваемых методу `.each()`, является функция обратного вызова, видоизменение обрабатываемых элементов, выбранных в объекте `jQuery`, не составляет труда. Добавим в дополнение `dateZoom` обработчики событий `hover` для каждого из элементов текущего выбранного набора.

При наведении указателя мыши на элемент, к которому применено дополнение `dateZoom`, будет выполняться метод `zoom`. Этот метод использует значение свойства `fontsize` объекта `defaults` для соответственного увеличения размера шрифта. При смещении указателя мыши в сторону методу `zoom` передается исходный размер шрифта, и размер текста элементов восстанавливается до первоначального значения.

Для хранения исходного размера шрифта, который можно получить с помощью метода `.css()`, используйте закрытую переменную.

Реализуйте описанную функциональность с помощью метода `.hover()`, введя в дополнение `dateZoom` следующий код, выделенный полужирным шрифтом.

```
(function($){

    // Дополнение, которое увеличивает размер текста элемента
    // при наведении на него указателя, а затем возвращает его
    // к исходному размеру при смещении указателя в сторону
    $.fn.dateZoom = function(options)
    {
        // Переопределить лишь те значения, которые были явно
        // переданы пользователем в параметрах
        var opts = $.extend($.fn.dateZoom.defaults, options);

        // Организовать цикл по всем выбранным элементам
        // и вернуть модифицированный объект jQuery для
        // поддержки возможности вызова в цепочках
        return this.each(function(){
            // Сохранить первоначальный размер шрифта элемента
            var originalsize = $(this).css("font-size");

            // Привязать функции к событию hover. Первая из них
            // запускается при наведении указателя мыши на элемент,
            // а вторая -- когда указатель смещается в сторону
            $(this).hover(function(){
                    $.fn.dateZoom.zoom(this, opts.fontsize, opts);
                },
                function(){
                    $.fn.dateZoom.zoom(this, originalsize, opts);
                });
        });
    };

    // Определить значения по умолчанию для дополнения
    $.fn.dateZoom.defaults = {
            "fontsize" : "110%",
            "easing" : "swing",
            "duration" : "600",
            "callback" : null
        };
```

```
    // Определить вспомогательную функцию, к которой пользователь
    // при необходимости сможет получить доступ извне дополнения
    $.fn.dateZoom.zoom = function(element, size, opts)
    {

        $(element).animate({
                    "font-size" : size
                },{
                    "duration" : opts.duration,
                    "easing" : opts.easing,
                    "complete" : opts.callback
                })
            .dequeue() // Предотвратить скачки анимации
            .clearQueue(); // Обеспечить выполнение только
                           // одной анимации
    };

})(jQuery);
```

## Внедрение дополнения

Разработка дополнения закончена, и теперь его можно внедрить в приложение. Для этого достаточно включить в приложение файл дополнения и выбрать набор элементов, к которым оно должно быть применено.

### Включение файла дополнения в приложение

Чтобы включить в приложение файл дополнения, потребуется видоизменить файл `footer.inc.php`, добавив в него новый дескриптор сценария. Как и в случае дополнения `validDate`, чтобы метод был доступен для вызова, дополнение `dateZoom` следует включить *перед* файлом `init.js`.

```
    <script type="text/javascript"
        src="http://www.google.com/jsapi"></script>
    <script type="text/javascript">
        google.load("jquery", "1");
    </script>
    <script type="text/javascript"
          src="assets/js/jquery.validDate.js"></script>
    <script type="text/javascript"
          src="assets/js/jquery.dateZoom.js"></script>
    <script type="text/javascript"
          src="assets/js/init.js"></script>
</body>

</html>
```

### Инициализация дополнения для набора элементов

Дополнение включено в приложение, так что теперь можно вызвать метод `.dateZoom()` для набора элементов. Изменения затронут файл `init.js`, поэтому откройте его сейчас.

Сначала измените заданное по умолчанию значение `fontsize` на `13px`, а затем добавьте метод `.dateZoom()` в цепочку, построенную на наборе элементов, выбранных

строкой "li a". Затем измените содержимое файла init.js, введя в него следующий код, выделенный полужирным шрифтом:

```
jQuery(function($){

var processFile = "assets/inc/ajax.inc.php",
    fx = {...}

// Установить размер шрифта по умолчанию для метода dateZoom
$.fn.dateZoom.defaults.fontsize = "13px";

// Захватывать события в модальном окне
// и присоединить эффект zoom effect
$("li a")
    .dateZoom()
    .live("click", function(event){

            // Предотвратить загрузку файла view.php
            // по щелчку на ссылке
            event.preventDefault();

            // Добавить в ссылку класс "active"
            $(this).addClass("active");

            // Получить строку запроса из атрибута href ссылки
            var data = $(this)
                        .attr("href")
                        .replace(/.+?\?(.*)$/, "$1"),

            // Проверить существование модального окна и выбрать
            // его или создать новое окно
            modal = fx.initModal();

            // Создать кнопку для закрытия окна
            $("<a>")
                .attr("href", "#")
                .addClass("modal-close-btn")
                .html("&times;")
                .click(function(event){
                            // Удалить модальное окно
                            fx.boxout(event);
                        })
                .appendTo(modal);

            // Загрузить информацию о событии из БД
            $.ajax({
                    type: "POST",
                    url: processFile,
                    data: "action=event_view&" + data,
                    success: function(data){
                        // Отобразить информацию о событии
                        fx.boxin(data, modal);
                    },
                    error: function(msg) {
```

```
                        alert(msg);
                    }
                });
    });

$(".admin-options form,.admin")
    .live("click", function(event){...});

// Редактировать события без перезагрузки страницы
$(".edit-form input[type=submit]")
    .live("click", function(event){...});

$(".edit-form a:contains(отменить)")
    .live("click", function(event){...};

});
```

Сохраните изменения, перезагрузите страницу `http://localhost/` в браузере и наведите указатель мыши на название события, чтобы увидеть дополнение `dateZoom` в действии (рис. 10.1).

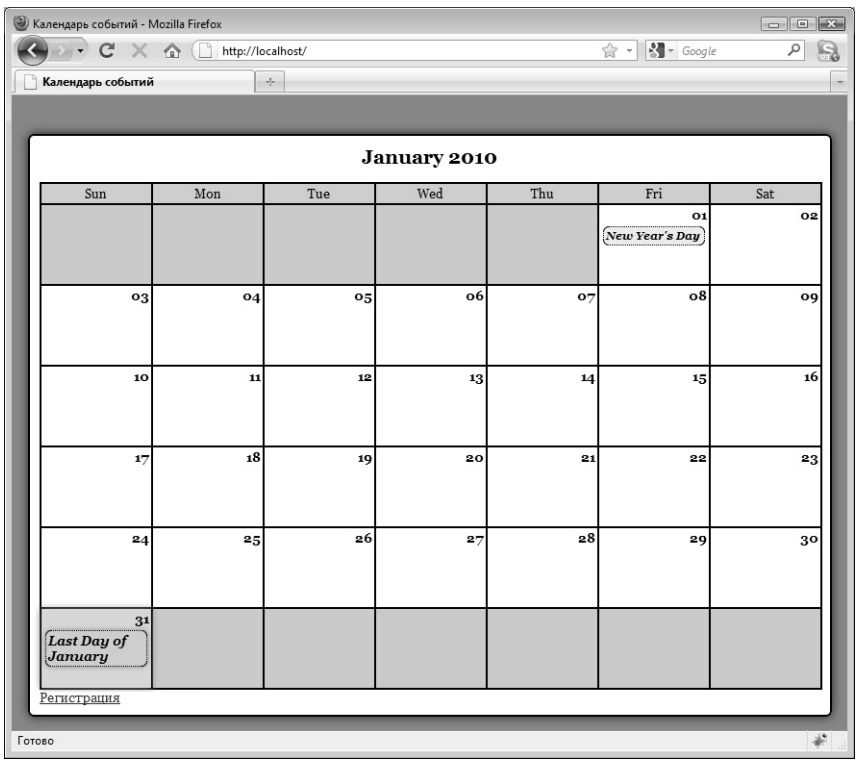



**Рис. 10.1.** Увеличение размера шрифта в названии события при наведении на него указателя мыши

# Резюме

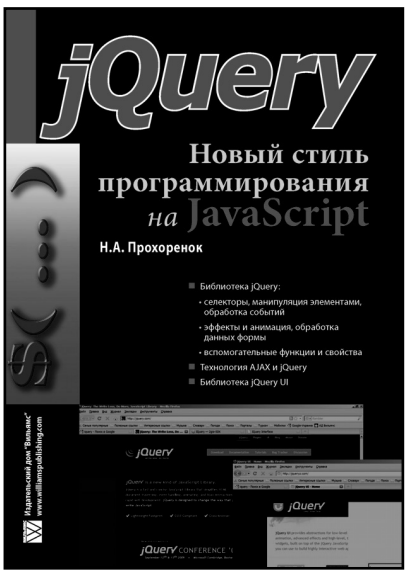
Теперь при создании пользовательских дополнений в jQuery, как в виде методов, пригодных для включения в цепочки, так и в виде добавляемых функций, вы должны чувствовать себя уверенно. Эта глава была довольно короткой, но именно эта краткость является лучшим свидетельством того, насколько просто можно расширить библиотеку jQuery собственными сценариями.

Примите поздравления! Вы научились использовать комбинацию PHP и jQuiery для создания пользовательских веб-приложений, по своему поведению приближающихся к настольным. Теперь вы полностью подготовлены к реализации своих замечательных идей во Всемирной сети!

# Предметный указатель

# JQUERY
## Новый стиль программирования на JavaScript

***Н.А. Прохоренок***




**www.williamspublishing.com**

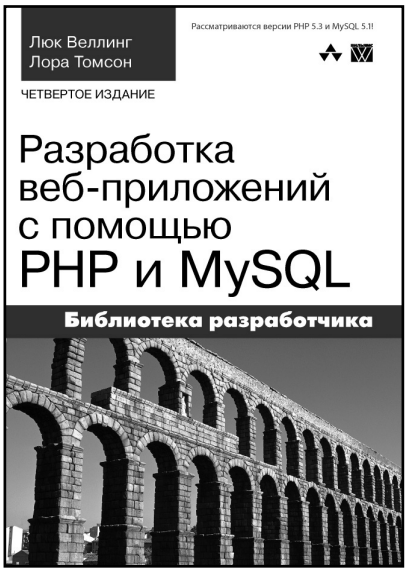
Книга является справочником по JavaScript-библиотеке jQuery. Рассматриваются функциональные возможности библиотеки, полезные для максимально широкого круга задач, включая механизм селекторов, манипулирование параметрами и содержимым элементов DOM-модели документа, обработку событий и данных форм. Продемонстрированы возможности использования технологии AJAX для обмена данными с сервером без перезагрузки страницы. Описаны как базовые свойства и методы объекта XMLHttpRequest, так и интерфейс доступа к AJAX, предоставляемый библиотекой jQuery. Кроме того в книге рассматривается библиотека визуальных компонентов jQuery UI, предоставляющая готовые решения, которые может использовать любой разработчик, даже не владея основами jQuery и JavaScript. Эта библиотека позволяет создавать в документе нестандартные компоненты, панели с вкладками и др.

**ISBN 978-5-8459-1603-7**

**в продаже**

# РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ PHP И MYSQL
## БИБЛИОТЕКА РАЗРАБОТЧИКА
## ЧЕТВЕРТОЕ ИЗДАНИЕ

***Люк Веллинг***
***Лора Томсон***




**www.williamspublishing.com**

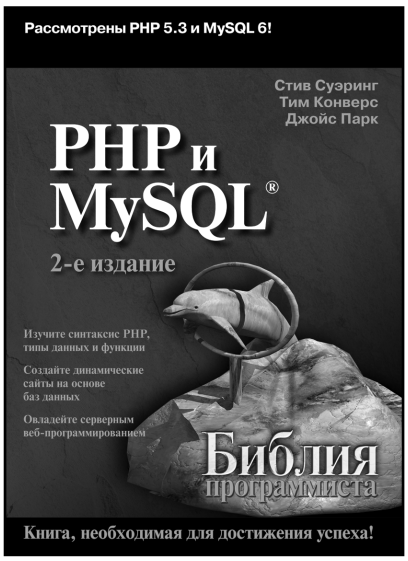
Эта книга предназначена для тех, кто знаком с основами HTML и ранее разрабатывал программы на современных языках программирования, но, возможно, не занимался программированием для веб или не использовал реляционные базы данных. В ней подробно описано применение последних версий PHP и MySQL для построения крупных коммерческих веб-сайтов. Основное внимание в книге уделено реальным приложениям. Здесь рассматриваются как простые интерактивные системы приема заказов, так и различные аспекты электронных систем продажи и безопасности во взаимосвязи с созданием реального веб-сайта. Подробно описаны все стадии разработки типовых проектов на PHP и MySQL, в числе которых служба веб-почты, приложение поддержки веб-форумов и электронный книжный магазин. Книга ориентирована на профессиональных разработчиков, но будет полезной и начинающим.

**ISBN 978-5-8459-1574-0**

**в продаже**

# PHP И MYSQL®
# БИБЛИЯ ПРОГРАММИСТА
## *2-е издание*

***Стив Суэринг,***
***Тим Конверс,***
***Джойс Парк***




В книге приведены исчерпывающие сведения по созданию динамических Web-сайтов на основе бесплатных программных средств с открытым исходным кодом (языка PHP, сервера Apache и СУБД MySQL), а также показано, как обеспечить бесперебойную эксплуатацию таких сайтов под управлением операционной системы Windows или Linux. Многочисленные сценарии и готовые программы, представленные в книге, подробно описаны, тщательно прокомментированы и составляют основу практически значимых приложений.

Книга дополняет оперативную документацию, содержит все необходимые справочные данные и рассчитана на широкий круг читателей.

**www.dialektika.com**

**ISBN 978-5-8459-1640-2** **в продаже**

# PHP: объекты, шаблоны и методики программирования, 2-е издание

***Мэтт Зандстра***


**www.williamspublishing.com**

В этой книге изложены методики объектно-ориентированного программирования и проектирования с точки зрения программиста на PHP.

Книга начинается с обзора объектно-ориентированных возможностей PHP, в который включены важные темы, наподобие определения класса, создания объектов, наследовании, инкапсуляции методов и свойств. Вы изучите также и дополнительные темы, такие как статические методы и свойства, абстрактные классы, обработка исключений, клонирование объектов и много другое.

Следующая часть книги посвящена шаблонам проектирования, которые органически дополняют тему ООП. В ней описываются концепции шаблонов проектирования и показаны способы реализации нескольких важных шаблонов в приложениях на PHP.

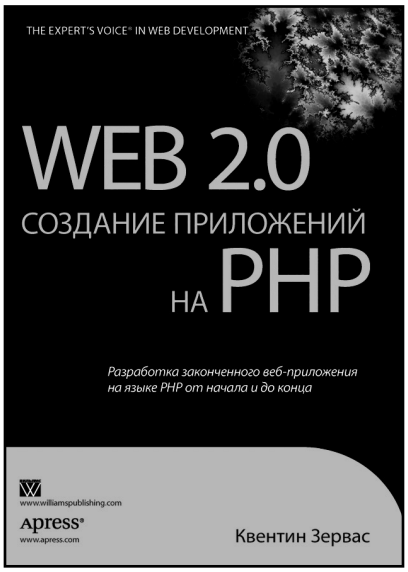
В последней части книги описывается несколько важных утилит, облегчающих процесс создания технической документации, управления работой групп программистов, тестирования кода и развертывания PHP-приложений: Phing, PHPUnit2, PHPDocumentor, PEAR и CVS.

**ISBN 978-5-8459-1586-3**

**в продаже**

# WEB 2.0 СОЗДАНИЕ ПРИЛОЖЕНИЙ НА PHP

***Квентин Зервас***




**www.williamspublishing.com**

Книга посвящена пошаговой разработке законченного практического веб-приложения, следующего стандартам Web 2.0 — многопользовательской системы блогов с возможностью размещения фотографий и географических карт. Разработка ведется на языке PHP с использованием библиотеки Zend Framework и системы Smarty Template Engine. Для реализации стандартных функций веб-сайтов и визуальных эффектов привлекаются библиотеки JavaScript-кода Prototype и Scriptaculous. Рассмотрено создание базы данных MySQL или PostgreSQL для обслуживания приложения, а также применение микроформатов и технологии Ajax.

Книга предназначена для студентов и специалистов в области веб-программирования сайтов с динамическим контентом.

**ISBN 978-5-8459-1590-0** **в продаже**

# МОДЕРНИЗАЦИЯ И РЕМОНТ ПК
## *19-е издание*

***Скотт Мюллер***


**www.williamspublishing.com**

В книге описаны все актуальные изменения в индустрии аппаратных средств ПК: новые процессоры Core i от Intel и Phenom от AMD, накопители SSD и новые модели жестких дисков, новые формфакторы материнских плат, новые наборы микросхем, новые блоки питания, память DDR3 и средства поддержки Windows 7, а также многое другое.

Книга будет чрезвычайно полезна как профессионалам в компьютерной отрасли, так и домашним пользователям, которые найдут здесь ответы на все вопросы.

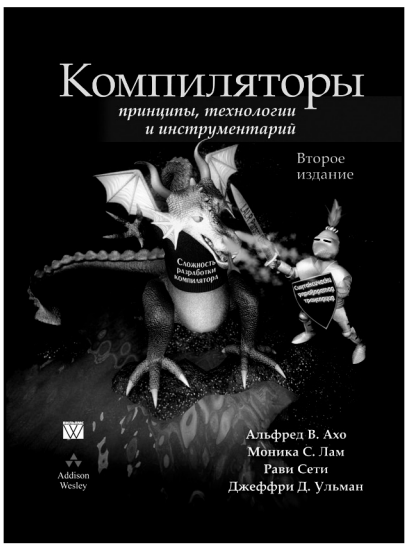
В этом издании:

- подробное рассмотрение новейших процессоров от Intel и AMD, в том числе семейств Intel Core i и AMD Phenom;
- обзор новых формфакторов материнских плат DTX и Mini ITX;
- расширенный обзор новых графических процессоров, а также наборов микросхем от NVIDIA и ATI/AMD;
- знакомство с новыми терабайтовыми жесткими дисками, накопителями SSD, а также другими инновационными технологиями хранения данных.

**ISBN 978-5-8459-1668-6** **в продаже**

# КОМПИЛЯТОРЫ
## ПРИНЦИПЫ, ТЕХНОЛОГИИ И ИНСТРУМЕНТАРИЙ
## 2-е издание

***Альфред В. Ахо,***
***Моника С. Лам,***
***Рави Сети,***
***Джеффри Д. Ульман***




**www.williamspublishing.com**

Каждый, кто интересовался разработкой компиляторов, не мог не слышать о знаменитой “Книге Дракона”, классическом труде Ахо и Ульмана “Принципы разработки компиляторов”. Развитие технологий компиляции привело к рождению очередного “дракона” — книги “Компиляторы. Принципы, технологии и инструментарий”, — у которого теперь уже четыре автора, и каждый из них является высококлассным специалистом в данной области. Книга, как и ранее, начинается с изложенияй основных принципов разработки компиляторов, включая детальное рассмотрение лексического и синтаксического анализа и генерации кода. Особенностью данного издания является широкое освещение вопросов оптимизации кода, в том числе для работы в многопроцессорных системах.

**ISBN 978-5-8459-1349-4**

**в продаже**

# ФУНКЦИИ В MICROSOFT OFFICE EXCEL 2010

***Г.И. Сингаевская***

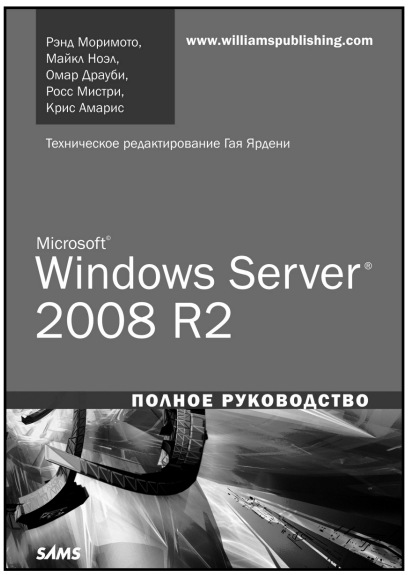

**www.dialektika.com**

Данная книга представляет собой полный курс по функциям Excel 2010. Здесь вы найдете подробное описание всех встроенных функций Excel 2010, в том числе и аналитических, с множеством примеров их использования в формулах. Последние главы книги посвящены разработке нестандартных функций для Excel. Достаточно много внимания в книге уделяется рассмотрению различных средств и инструментов Excel 2010, использование которых для решения ряда задач оказывается более эффективным, чем использование формул и функций. Изложение проиллюстрировано многочисленными примерами различного уровня сложности. Книга рекомендуется всем, кто желает максимально эффективно использовать Excel 2010 в своей повседневной деятельности.

**ISBN 978-5-8459-1669-3** **в продаже**

# MICROSOFT WINDOWS SERVER 2008 R2
## *Полное руководство*

***Рэнд Моримото,***
***Майкл Ноэл,***
***Омар Драуби,***
***Росс Мистри,***
***Крис Амарис***




**www.williamspublishing.com**

В книге приводится исчерпывающее описание всех аспектов Windows Server 2008 R2, включая Active Directory, сетевые службы, безопасность, переход на Windows Server 2008 R2 из Windows Server 2003/2008, администрирование, отказоустойчивость, оптимизация, обнаружение неполадок, ключевые прикладные службы и многое другое.

Авторы очень точно отмечают основные усовершенствования Windows Server 2008 R2 и предлагают развернутое описание всех инноваций Windows Server 2008 R2, начиная с виртуализации Hyper-V и заканчивая DirectAccess и улучшениями компонента Failover Clustering. В каждой главе предлагаются многочисленные полезные советы и трюки, основанные на сотнях внедрений, которые позволяют максимально эффективно решать бизнес-задачи с помощью Windows Server 2008 R2.

Книга рассчитана на пользователей и администраторов средней и высокой квалификации.

**ISBN 978-5-8459-1653-2** **в продаже**

# РАСКРУТКА ВЕБ-САЙТА
## ПРАКТИЧЕСКОЕ РУКОВОДСТВО

***Н.В. Евдокимов***
***И.В. Лебединский***


**www.williamspublishing.com**

Эта книга — совершенно уникальное явление в литературе по продвижению веб-сайтов. Нигде больше вы не найдете столь полного и в тоже время простого и понятного изложения как теоретических концепций, так и сугубо практических методов и рекомендаций. Процедура продвижения сайта раскрыта в этом издании во всей своей полноте — от планирования самого сайта, до мероприятий по контролю и корректировке достигнутых результатов. Самые современные понятия, технологии и подходы обсуждаются в строгой последовательности и предлагаются читателю к ежедневному применению в его практической деятельности. Твердая теоретическая база, подкрепленная “живыми”, взятыми из реальной жизни примерами, четкая структура, живой и образный язык — все это делает данное руководство настольной книгой как для начинающих, так и для опытных специалистов в области интернет-маркетинга.

**ISBN 978-5-8459-1679-2** **в продаже**

# HTML, XHTML И CSS БИБЛИЯ ПОЛЬЗОВАТЕЛЯ
## *5-е издание*

***Стивен Шафер***


**www.dialektika.com**

В этом исчерпывающем руководстве приводится подробное описание всех “строительных блоков” веб-страниц — языка гипертекстовой разметки (HTML), расширяемой версии HTML (XHTML) и каскадных таблиц стилей (CSS). Изучите основы написания сценариев и стандарты кодирования, работу с тегами, таблицами, формами и ссылками, научитесь тестировать и проверять код и освойте другие технологии, которые позволят вам стать настоящим профессионалом в области веб-программирования.

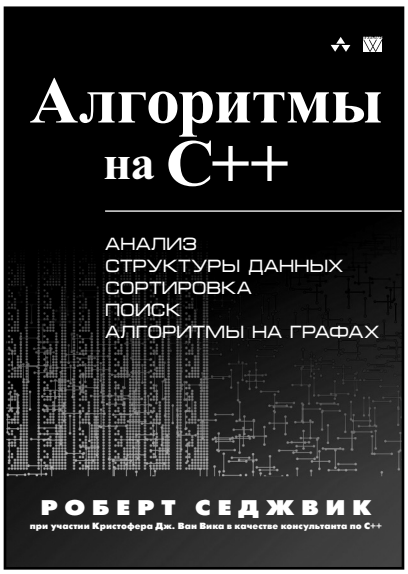
Освойте методы создания динамического контента с помощью мультимедийных объектов и сценариев.

Научитесь выполнять сложное форматирование веб-документов быстро и качественно.

**ISBN 978-5-8459-1676-1** **в продаже**

# АЛГОРИТМЫ НА C++
## АНАЛИЗ, СТРУКТУРЫ ДАННЫХ, СОРТИРОВКА, ПОИСК, АЛГОРИТМЫ НА ГРАФАХ

***Роберт Седжвик***




**www.williamspublishing.com**

Эта классическая книга удачно сочетает в себе теорию и практику, что делает ее популярной у программистов на протяжении многих лет. Кристофер Ван Вик и Роберт Седжвик разработали новые лаконичные реализации на C++, которые естественным и наглядным образом описывают методы и могут применяться в реальных приложениях. Каждая часть содержит новые алгоритмы и реализации, усовершенствованные описания и диаграммы, а также множество новых упражнений для лучшего усвоения материала. Акцент на АТД расширяет диапазон применения программ и лучше соотносится с современными средами объектно-ориентированного программирования. Книга предназначена для широкого круга разработчиков и студентов.

**ISBN 978-5-8459-1650-1**

**в продаже**

# C# 4.0
## ПОЛНОЕ РУКОВОДСТВО

***Герберт Шилдт***


**www.williamspublishing.com**

В этом полном руководстве по C# 4.0 — языку программирования, разработанному специально для среды .NET, — детально рассмотрены все основные средства языка: типы данных, операторы, управляющие операторы, классы, интерфейсы, методы, делегаты, индексаторы, события, указатели, обобщения, коллекции, основные библиотеки классов, средства многопоточного программирования и директивы препроцессора. Подробно описаны новые возможности C#, в том числе PLINQ, библиотека TPL, динамический тип данных, а также именованные и необязательные аргументы. Это справочное пособие снабжено массой полезных советов авторитетного автора и сотнями примеров программ с комментариями, благодаря которым они становятся понятными любому читателю независимо от уровня его подготовки. Книга рассчитана на широкий круг читателей, интересующихся программированием на C#.

**ISBN 978-5-8459-1684-6** **в продаже**