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).

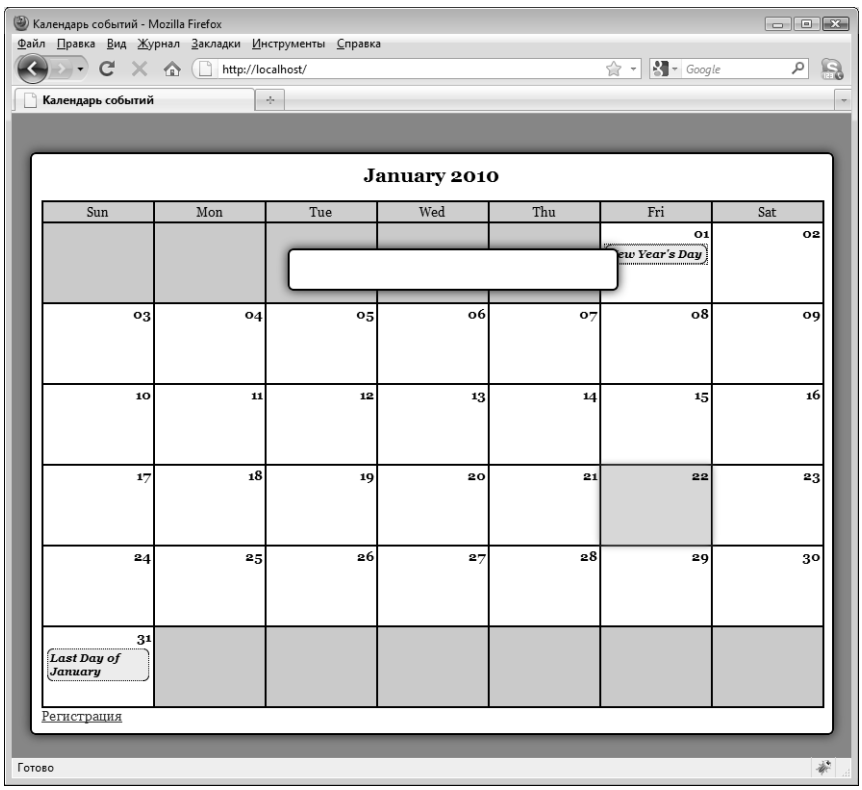

Рис. 7.1. Появление модального окна после щелчка на названии события

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).

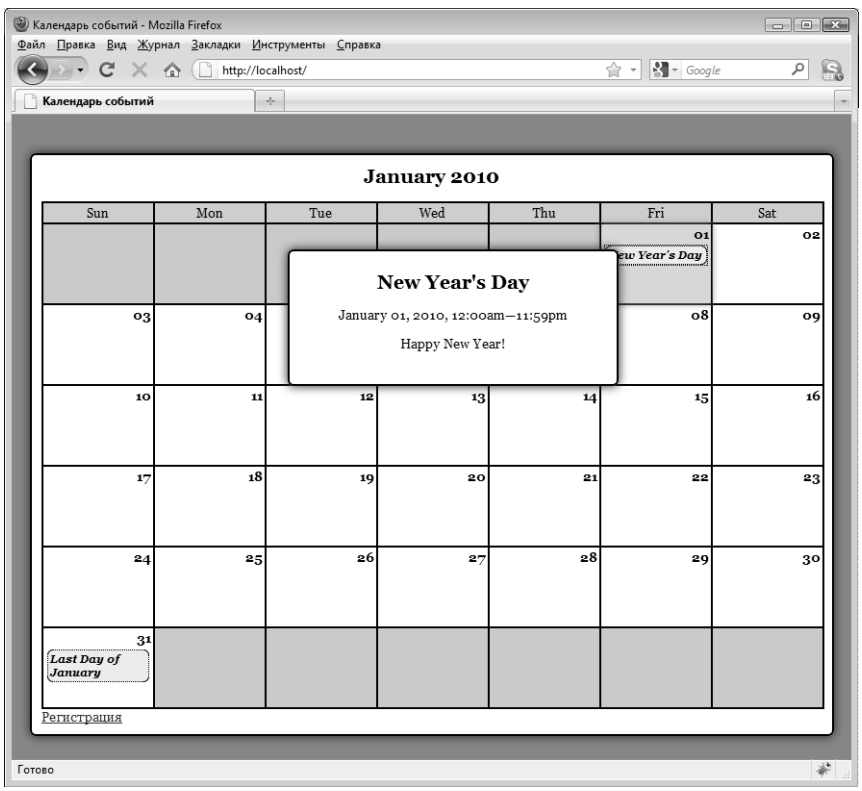

Рис. 7.2. Информация о событии загружена в модальное окно

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.

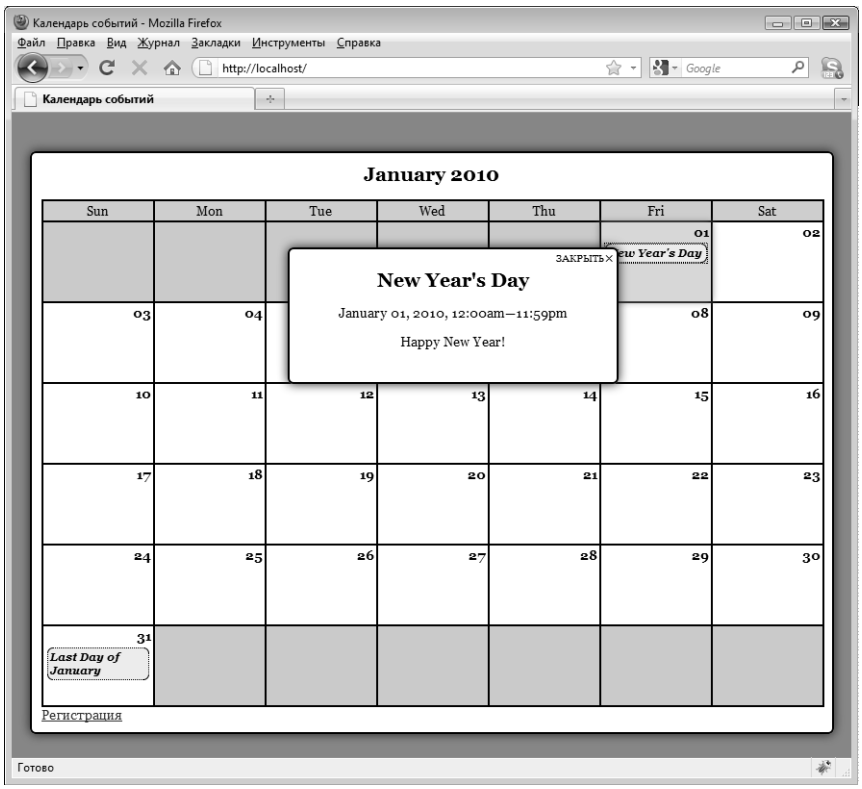

Рис. 7.3. Теперь в модальном окне отображается кнопка Закрыть

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).

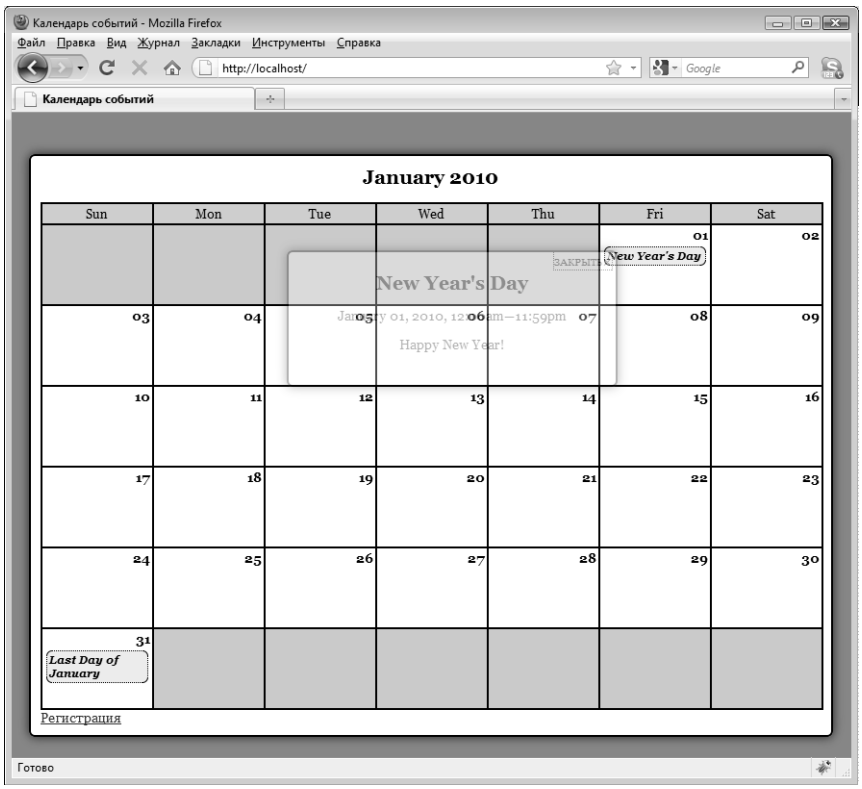

Рис. 7.4. Модальное окно в процессе плавного исчезновения после щелчка на кнопке Закрыть

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
        }

};
```

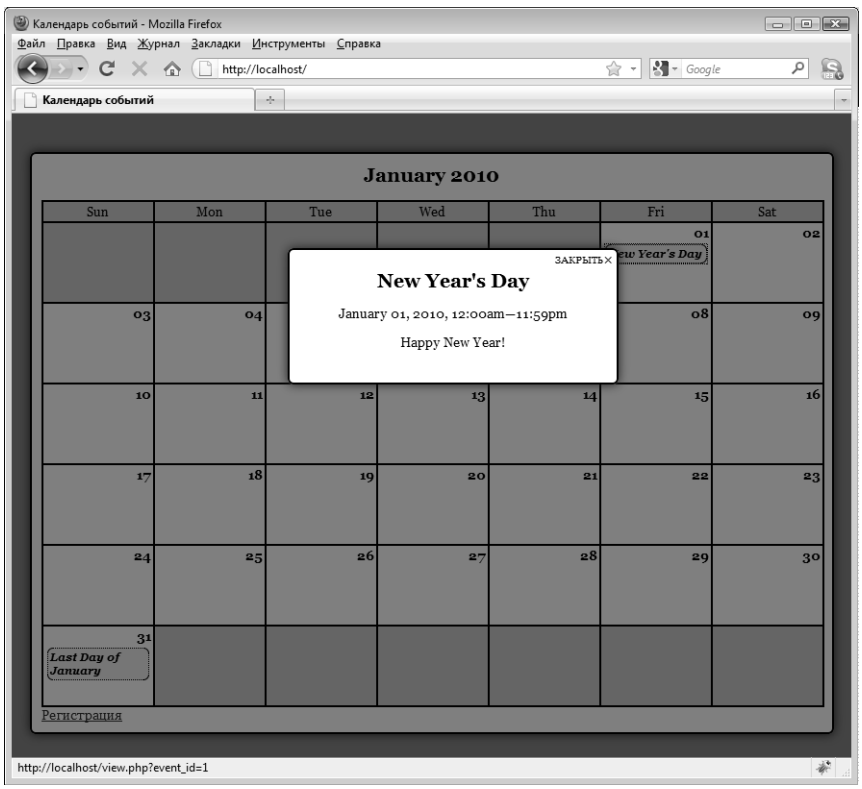

Рис. 7.5. Использование модального окна с оверлеем фокусирует внимание на отображаемой информации

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.