1. Лекция: Архитектура Joomla. Базовые сведения

Коротко описана архитектура Joomla. Рассмотрены предопределенные константы, языковые файлы, реализация паттерна "фабрика", работа с HTTP-запросом, объектом JApplication, создание панелей инструментов.

Цель лекции: Изучить базовые сведения об архитектуре Joomla. Получить представление о функционировании такого типа расширений как компоненты.

Предисловие

Важнейшим источником сведений о программировании под Joomla является официальная документация к этой системе [5]. Однако на момент составления данного курса эта документация является достаточно неполной, особенно с учетом того, что ряд статей относится к старой версии Joomla (1.5). Код расширений, написанных под Joomla 1.5, несколько отличается от кода, написанного под недавно вышедшие версии Joomla 1.6, 1.7 и 2.5.

Руководство по разработке компонента на официальном сайте Joomla начинается сразу с написания кода в соответствии с архитектурой MVC (Модель – Вид – Представление). Такой подход едва ли удобен, так как значительно легче для студента было бы начать с изучения основ программирования под Joomla, с изучения основных классов фреймворка этой системы и лишь потом переходить на архитектуру MVC. Более удобный для обучения подход применяется в книге [3]. Она строится от простого к сложному: сначала рассматривается разработка простого компонента без использования классов, реализующих MVC, а затем происходит переход на эту архитектуру. Поэтому в практической части данного курса мы будем следовать порядку изложения этой книги. Кроме того, большая часть программного кода, составляющего практическую часть данного курса, основана на листингах из этой книги. Однако исходные коды, взятые из книги [3], написаны под Joomla 1.5 и устарели. Для данного курса они были изменены для использования в Joomla 1.7 и перенесены на другую предметную область. Переход на Joomla 1.7 потребовал расмотрения в практической части курса таких отсутствовавших в [3] вопросов, как создание пунктов меню в панели управления, использование языковых файлов. Большее внимание уделено работе с навигационной цепочкой сайта.

Написание теоретической части лекций усложнялось пробелами в документации Joomla. Иногда описание параметров какого-либо метода присутствует в документации, но является совершенно неверным. В ряде случаев при рассмотрении классов фреймворка этой системы приходилось открывать исходный код ее файлов, чтобы разобраться, как работает тот или иной метод. Особенно часто этот подход применялся при рассмотрении группы классов, управляющих генерацией элементов HTML.

Следует отметить отсутствие литературы о программировании под Joomla на русском языке. Исключением является небольшое руководство [6], написанное тем же Джозефом ЛеБланком и опубликованное в русском переводе в электронном журнале PHPInside за ноябрь – декабрь 2005 г.

Отметим также книгу [2], которая тоже рассматривает программирование для старой версии

Joomla.

В начале 2012 г. вышла книга [1], восполняющая недостаток сведений о программировании под Joomla 1.6/1.7/2.5.

Введение

Для системы управления контентом Joomla созданы тысячи расширений. Тем не менее, использовать готовое решение не всегда целесообразно. Стороннее расширение может быть слишком дорогим или перегружать сервер ненужными для конкретной задачи функциями. Для нестандартной задачи готового решения может вовсе не найтись.

Иногда достаточно воспользоваться одним из конструкторов контента (CCK) для Joomla, позволяющих создавать свои шаблоны для материалов. Однако и CCK - не панацея, и возможно, что и он окажется бессилен. В таком случае возникает необходимость написать собственное расширение.

Может случиться, что готовое решение начнет работать некорректно и придется искать в нем ошибку. Тогда, чтобы разобраться в его коде, программисту понадобится знание принципов построения расширений под Joomla.

Возможно, необходим какой-нибудь модуль для готового расширения, например, вывод списка последних комментариев к фотографиям, но такого модуля для этого расширения нет. Может быть, отдельные части существующего компонента являются платными и слишком дорогими. В этих случаях также стоит задуматься о разработке собственного расширения.

Архитектура Joomla

Фреймворк Joomla состоит из трех уровней (рис. 1.1):

1. уровень фреймворка;
2. уровень приложения;
3. уровень расширений.

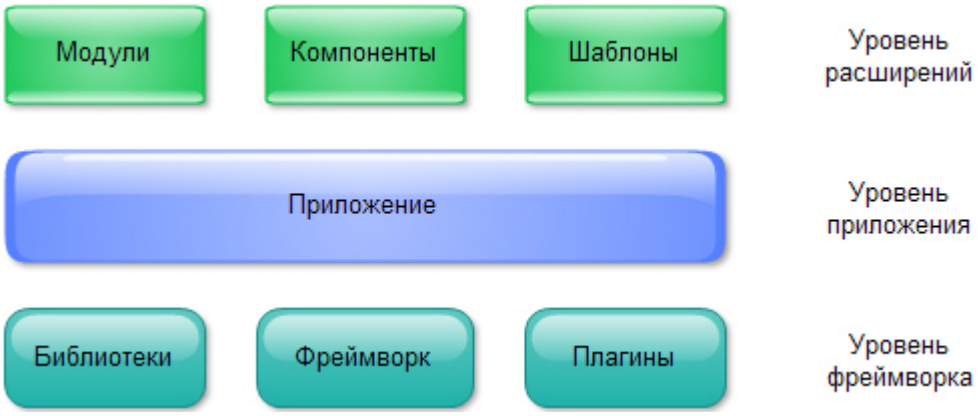


Рис. 1.1. Фреймворк Joomla

Уровень фреймворка обеспечивает базовую функциональность Joomla с помощью набора библиотек и плагинов и собственно фреймворка Joomla:

- **фреймворк Joomla** (или **"ядро"**) - набор классов, обеспечивающих базовую функциональность Joomla. Названия этих классов начинаются с буквы "J" и говорят сами за себя: JDatabase, JUser, JForm, JEditor и т.д.;
- **библиотеки** требуются для работы фреймворка или сторонних расширений;

- **плагины** расширяют функциональность фреймворка.

Уровень приложения состоит из приложений, которые расширяют абстрактный класс JApplication. **Приложение** - глобальный объект, использующийся для обработки запросов.

В этот уровень входят следующие приложения:

- JInstallation запускается при установке Joomla. После завершения установки необходимо удалить директорию installation, которая как раз и содержит данное приложение. В дальнейшем установка расширений выполняется с помощью приложения JAdministrator;
- JAdministrator управляет всеми функциями для администрирования Joomla;
- JSite отвечает за компоновку и отображение фронтенда;
- XML-RPC позволяет администрировать сайт Joomla удаленно.

Уровень расширений состоит из расширений фреймворка Joomla и приложений:

- **компоненты** - основной тип расширений Joomla. При каждом обращении к Joomla происходит вызов соответствующего компонента. Например, при отображении какой-либо страницы сайта происходит вызов компонента com_content;
- **модули** используются для отображения небольших фрагментов контента, обычно в левой или правой колонке или верхней или нижней областях страницы;
- **плагины** позволяют зарегистрировать функции и классы для обработки каких-либо событий, вызванных Joomla, например, поиск по сайту;
- **языковые файлы** позволяют представить контент Joomla на нескольких языках;
- **шаблоны** отвечают за внешний вид сайта.

Фронтенд и бэкенд

Joomla делится на **фронтенд** - часть сайта, доступная пользователю, и **бэкенд** - систему администрирования сайта. Соответственно, в Joomla всего две точки входа - index.php для фронтенда и /administrator/index.php для бэкенда. Чтобы вызвать какой-либо установленный на сайте компонент, необходимо передать его имя (с префиксом "com_") скрипту index.php или /administrator/index.php в переменной option в строке URL. Например, переход по ссылке http://localhost/joomla/index.php?option=com_banners приведет к вызову компонента banners.

Большинство компонентов для Joomla делятся на фронтенд и бэкенд, и их код распределяется по двум папкам, каждая из которых называется по схеме com_<имя компонента>. В каждой из этих папок должен находиться файл, являющийся точкой входа, и называющийся так же, как компонент, т.е. <имя компонента>.php. Схематически это можно изобразить так:

```
administrator
|- components
..|- com_mycomponent
....|- mycomponent.php
components
..|- com_mycomponent
....|- mycomponent.php
```

Предопределенные константы

В Joomla определен ряд констант, хранящих значения путей: JPATH_BASE - путь к корневой директории текущего приложения; JPATH_ROOT - путь к корневой директории сайта,

JPATH_COMPONENT - путь к директории компонента, JPATH_COMPONENT_SITE - путь к фронтенду компонента, JPATH_COMPONENT_ADMINISTRATOR - путь к бэкенду компонента и т.д. Полный их список можно найти в документации. Все эти константы возвращают значения абсолютных путей в файловой системе. Если вам необходимо получить путь для использования в URL, следует воспользоваться методом JURI::base().

В файле index.php, расположенном в корневой директории Joomla, определена константа _JEXEC. Большинство PHP-файлов, написанных под Joomla, начинаются с выражения

```
defined('_JEXEC') or die('Restricted access');
```

Данное выражение осуществляет проверку, был ли файл, в котором оно записано, вызван из Joomla. Таким путем запрещается доступ к файлу извне, чтобы предотвратить взлом сайта.

Еще одна популярная константа Joomla - DS, разделитель директорий, принятый в конкретной операционной системе (например, прямой или обратный слеш).

Языковые файлы

Joomla позволяет создать мультиязыковый сайт, задавая для каждого пользователя язык сайта и панели управления. Данная возможность реализована следующим образом: в кодах расширений при необходимости вывести на экран какой-либо заранее известный текст (например, сообщение об успешном выполнении запроса пользователя) вместо этого текста записывается его эквивалент (**ключ**). Для каждого языка, поддерживающегося данным расширением, создаются языковые файлы, которые хранят **переводы** для всех ключей, встретившихся в кодах расширения. Например, для ключа "COM_MYCOMPONENT_HELLO_WORLD" перевод на английский язык может задаваться как "Hello, world!", на русский - "Здравствуй, мир!", на французский - "Bonjour le monde!" и т.д.

Языковые файлы фронтенда хранятся в папке **/language/<ln-LN>**, где <ln-LN> - код языка по стандарту RFC3066. Файл должен называться по схеме <ln-LN>.<префикс><имя расширения>.ini, где префикс зависит от вида расширения: "com_" (компонент), "mod_" (модуль), "tpl_" (шаблон) и т.д. Например, путь к языковому файлу компонента contact для русского языка следующий:**/language/ru-RU/ru-RU.com_contact.ini**

Языковые файлы бэкенда хранятся в папке **/administrator/language/<ln-LN>**.

Кроме файлов .ini, для расширения должен также быть создан файл *.sys.ini, в котором могут храниться переводы сообщений, выводящихся после установки расширения, переводы пунктов меню, создающихся для компонента в панели управления, переводы параметров компонента и переводы надписей, выводящихся в менеджере расширений. Например, путь к файлу .sys.ini компонента contact для русского языка выглядит так: **/administrator/language/ru-RU/ru-RU.com_contact.sys.ini**

Содержимое языкового файла состоит из пар "ключ-значение" и, при необходимости, комментариев. Пустые строки игнорируются. Комментарии начинаются с символа ";". Например

```
; Это комментарий
```

Ключ - это строка для перевода, а значение - это перевод данной строки на заданный язык. Ключ отделяется от значения знаком равенства:

```
КЛЮЧ=Значение
```

Например:

```
COM_CONTACT="Контакты"
```

Ключ должен быть записан в верхнем регистре и не должен содержать пробелы. Все ключи во фронтенде должны начинаться со строки <префикс><имя расширения>_, например:

```
COM_CONTACT_CHANGE_CONTACT_BUTTON
```

Значение (перевод ключа) должно быть заключено в двойные кавычки. Если значение к тому же содержит двойные кавычки, то они должны быть записаны в виде HTML-сущности, например, &quot.

Для использования переводов применяются методы статического класса JText _(), sprint() и printf().

Простейший способ вывести перевод строки - использовать метод JText::_(), который просто переводит строку, переданную ему параметром. Например:

```
echo JText::_('COM_MYCOMPONENT_HELLO_WORLD');
```

Если перевод для заданной строки не будет найден в языковых файлах, то метод _() вернет саму эту строку. Перед поиском строка будет переведена в верхний регистр, поэтому не имеет значения, в каком регистре ее записать в коде.

Если в строку необходимо включить какие-либо значения, то используются методы JText sprintf() и printf(), аналогичные одноименным функциям в PHP. Их параметрами являются строка для перевода и любое количество аргументов для подстановки в переведенную строку. Сами параметры не будут переведены. Методы sprintf() и printf() различаются тем, что printf() выводит получившуюся строку на экран и возвращает ее длину, а sprintf() возвращает саму строку и ничего не выводит.

Например, если в языковом файле ru-RU.com_mycomponent.ini задано

```
COM_MYCOMPONENT_THANK_YOU="Спасибо за Ваше сообщение, %s!"
```

а в коде расширения имеется строка

```
echo JText::sprintf('COM_MYCOMPONENT_THANK_YOU', 'Вася');
```

то результатом будет вывод на экран строки "Спасибо за Ваше сообщение, Вася!".

Аргументы задаются так же, как в одноименных функциях PHP: %s означает строку, %d - целое число, %f - число с плавающей точкой и т.д.

Паттерн "фабрика" (класс JFactory)

В Joomla существует статический класс JFactory, реализующий паттерн "фабрика". Методы данного класса (getApplication(), getDate(), getDbo(), getDocument(), getLanguage(), getURI(), getUser(), getMailer(), getEditor() и др.) позволяют получить доступ к соответствующим

глобальным объектам фреймворка (JApplication, JDate, JDatabase, JDocument, JLanguage, JURI, JUser, JMail, JEditor и др.), ряд которых будет рассмотрен далее.

Рассмотрим пример получения доступа к объекту JUser:

```
$user =& JFactory::getUser();
if ($user->guest)
  echo "Вы не вошли на сайт";
else
  echo "Вы вошли на сайт как ".$user->name;
```

Обратите внимание на знак амперсанда перед вызовом метода getUser(). Мы получаем ссылку на объект-представитель текущего пользователя. Если пропустить амперсанд, то будет создана копия этого объекта и изменения, которые мы будем в ней производить, не затронут оригинал.

HTTP-запрос (класс JRequest)

В Joomla вместо непосредственного использования глобальных массивов **$_GET, $_POST, $_FILES, $_COOKIE, $_ENV, $_SERVER и $_REQUEST** удобнее применять класс **JRequest**. Его методы пропускают данные, введенные пользователем, через фильтр во избежание инъекций.

Для получения переменных запроса GET/POST используется метод **mixed getVar(string $name, string $default=null, string $hash='default', string $type='none', int $mask=0)**, где:

$name	имя переменной;	
$default	значение по умолчанию, которое вернет метод getVar(), если значение переменной не задано;	
$hash	источник данных, по умолчанию они будут получены из массива $_REQUEST. Явное указание массива GET или POST повысит безопасность кода;	
$type	тип ожидаемого значения:	
	INT	
	INTEGER	
	FLOAT	
	DOUBLE	
	BOOL	
	BOOL	
	WORD	
	ALNUM	допускает только буквенно-цифровые значения;
	CMD	
	BASE64	допускает только те символы, которые могут быть представлены в кодировке base64 (т.е. a-z, A-Z, 0-9, /, + и =);
	STRING	

	ARRAY	
	PATH	исключает возможность атаки. Например, если исходное значение содержало /./ или /../, то вместо него метод вернет пустую строку;
	USERNAME	удаляет управляющие символы (0x00 - 0x1F), 0x7F, <, >, ", ', % и &;
$mask		
	константа, задающая опции фильтрации:	
	JREQUEST_NOTRIM	не удалять пробелы в начале и конце строки;
	JREQUEST_ALLOWRAW	без какой-либо фильтрации;
	JREQUEST_ALLOWHTML	не удалять HTML-код, но пропустить значение через фильтр (в частности, удалить опасные теги - script, applet, iframe и др.).

Эти константы не заключаются в кавычки, т.к. это не строки, а статические переменные. Если ни одной опции не задано, то HTML-теги, а также пробелы в начале и конце строки будут удалены.

Пример:

```
$answer = JRequest::getVar('answer', 'no answer', 'post', 'string', JREQUEST_ALLOWRAW);
```

Если нужно получить весь массив переменных запроса в отфильтрованном виде, используется

```
mixed get(string $hash='default', int $mask=0)
```

Например, получим массив $_POST:

```
$arr = JRequest::get('post');
```

Для присвоения переменным запроса значений используется метод string setVar(string $name, string $value=null, string $hash='method', bool $overwrite=true)

Если $overwrite=false и в запросе уже задано значение переменной $name, то метод просто вернет само это значение. В противном случае переменной будет присвоено значение $value, а метод вернет старое значение $name.

Пример:

```
JRequest::setVar('var1', 'val1');
```

Класс JRequest содержит также методы, позволяющие получить значение определенного типа: **getBool(), getCmd(), getFloat(), getInt(), getString(), getWord()**.

Приложение (класс JApplication)

Очередь сообщений

В Joomla существует **очередь сообщений** - массив строк, которые будут выведены на экран при следующей загрузке какой-либо страницы. Стандартными являются три типа сообщений (рис. 1.2): message (собственно сообщение), notice (предупреждение) и error (ошибка):

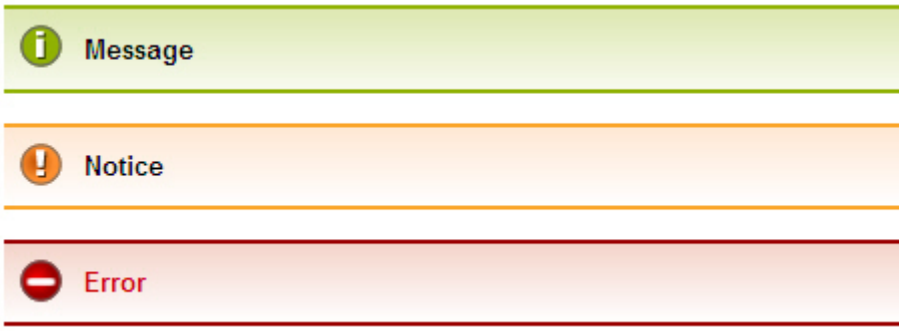


Рис. 1.2. Типы сообщений

Для добавления сообщений в очередь используется метод **void enqueueMessage(string $msg, [string $type = 'message'])**, где:

$msg - текст сообщения;
$type - тип сообщения.

Например:

```
global $app;
$app->enqueueMessage('Message');
$app->enqueueMessage('Notice', 'notice');
$app->enqueueMessage('Error', 'error');
```

В данном примере $app - это глобальный объект JApplication.

Для получения копии очереди сообщений используется метод array getMessageQueue(). Например, для предыдущей очереди из трех сообщений он возвращает массив:

```
Array
(
  [0]=>Array
   (
    [message]=>Message
    [type]=>message
  )
  [1]=>Array
  (
    [message]=>Notice
    [type]=>notice
  )
  [2]=>Array
  (
    [message]=>Error
```

```
    [type]=>error
  )
)
```

Перенаправление

Для перенаправления пользователя к другому URL используется метод void redirect(string **$url, string $msg='', string $msgType='message', bool $moved=false)**, где:

$url	- URL, к которому перенаправляется пользователь;
$msg	- сообщение, которое должно быть при этом выведено;
$msgType	- тип сообщения;
$moved	- при значении true браузер получит код состояния "301 Permanently Moved", в противном случае - "303 See Other".

Данный метод добавляет сообщение к очереди сообщений, перенаправляет браузер пользователя к заданному URL и завершает работу приложения Joomla. Например:

```
global $app;
$app->redirect('index.php', JText::_('NOTICE'), 'notice');
```

Второй способ организации перенаправления - использовать метод JController::setRedirect(), который будет рассмотрен ниже вместе с другими методами класса JController.

Получение параметров конфигурации сайта

В число параметров конфигурации сайта входят настройки базы данных, почты, сервера, FTP, метаданных, SEO и другие. Для получения значений этих параметров используется метод

```
mixed getCfg(string $varname, string $default=null)
```

где $varname - название параметра.

Для примера получим название сайта:

```
global $app;
echo $app->getCfg('sitename');
```

Определение типа запущенного приложения Joomla

Чтобы определить, откуда запущен код, можно использовать методы

int
getClientId()
bool isAdmin()

bool isSite()

Метод getClientId() возвращает id запущенного приложения: 0 (сайт), 1 (панель управления), 2 (установщик).

Метод isAdmin() определяет, является ли запущенное приложение бэкендом, isSite() - фронтендом.

Панели инструментов (класс JToolBarHelper)

Joomla автоматически загружает в верхней правой части экрана бэкенда компонента файл, который называется toolbar.<имя компонента>.php. Таким образом можно отображать различные панели инструментов.

Класс JToolBarHelper содержит методы, которые генерируют HTML-код для построения кнопок панелей инструментов. Для отображения кнопок, которые часто используются в компонентах, - "Сохранить", "Отменить", "Удалить" - существуют готовые методы этого класса. Их список можно найти в документации Joomla: http://docs.joomla.org/JToolBarHelper.

Для методов addNew(), publish(), publishList(), makeDefault(), unpublish(), editList(), save(), apply() заданы по умолчанию два параметра - задача (об ее значении будет сказано ниже) и подпись. Например, значения этих параметров по умолчанию для метода editList() выглядят так: $task = 'edit', $alt = 'Edit'. Можно задавать свои задачу и подпись, передавая их как, соответственно, первый и второй параметры.

Метод для удаления объектов имеет прототип

```
void deleteList(string $msg =, string $task = 'remove', string $alt =
'JTOOLBAR_DELETE')
```

где $msg - это текст сообщения с просьбой подтвердить удаление объектов. Если этот параметр задан, то при нажатии кнопки для удаления будет выводиться окно с заданным сообщением и объекты будут удалены только после нажатия кнопки "**ОК**" в этом окне. Если же параметр $msg не задан, то объекты будут удаляться без предупреждения.

В сложных случаях, когда требуется не просто изменить задачу и подпись, а создать собственную кнопку, используется метод

```
void custom(string $task = '', string $icon = '', string $iconOver = '', string
$alt = '', bool $listSelect = true)
```

где:

$task	- задача, которая будет выполнена;
$icon	- пиктограмма кнопки;
$iconOver	- пиктограмма при наведении курсора мыши;
$alt	- подпись под кнопкой;
$listSelect	- нужно ли работать только с выбранными элементами списка.

Как правило, в URL Joomla присутствует переменная task, определяющая задачу, которую

должен выполнить компонент. В коде компонента в зависимости от полученного значения task вызывается некоторая функция. Например, если URL выглядит как http://localhost/joomla/index.php?option=mycomponent&task=show, то компонент mycomponent будет обрабатывать задачу show.

Параметры $icon и $iconOver задают, как ни странно, не название файла изображения, а название класса CSS, для которого задано это изображение в качестве фонового. К названию класса автоматически добавится строка "icon-32-" и будет произведен поиск этого класса в подключенных файлах CSS. Например, если третий параметр функции JToolBarHelper::custom() задан как send.png, то будет найден класс .icon-32-send, а в результате картинка будет отображена с помощью кода:

```
<span class="icon-32-send"></span>
```

Если данная панель инструментов создана для одной-единственной записи, а не для списка, то параметру $listSelect следует задавать значение false. Если этот параметр имеет значение true, то для события кнопки onclick задается следующий код на Javascript:

```
if (document.adminForm.boxchecked.value==0)
{
  alert('Пожалуйста, выберите объект из списка');
}
else
{
  Joomla.submitbutton('myquestions_sendToExpert')
}
```

Если же $listSelect имеет значение false, то проверка того, выбраны ли в списке какие-либо элементы, не осуществляется:

```
Joomla.submitbutton('myquestions_sendToExpert')
```

Для вывода названия панели инструментов и пиктограммы служит метод void title(string $title, string $icon), где:

$title	- название панели инструментов;
$icon	- название класса CSS, для которого необходимое изображение задано в качестве фонового. К названию класса автоматически добавится строка "icon-48-".

Например, если вы хотите использовать в качестве пиктограммы файл **/media/com_mycomponent/images/sample-48.png**, то добавьте в CSS класс

```
.icon-48-sample
{
  background: url('../images/sample-48.png') 0 0 no-repeat;
}
```

Теперь для отображения названия необходимо добавить в код строку:

```
JToolBarHelper::title('Мой компонент','sample');
```

Для вывода между кнопками вертикальной черты-разделителя служит метод divider():

```
void divider();
```

Практика

Описание учебного примера

Для примера будем создавать компонент системы "вопрос - ответ". Назовем его **myquestions**. С помощью этого компонента посетители сайта смогут задавать свои вопросы, по желанию помечая их как предназначенные для публикации на сайте или как скрытые. Функциональность, доступная администратору системы:

- отправить уведомление о вопросе эксперту по электронной почте;
- задать дату снятия вопроса с публикации;
- скрыть от посетителей сайта отдельные поля вопроса;
- присвоить вопросу категорию;
- редактировать список категорий;
- ответить на вопрос;
- отправить ответ автору вопроса по электронной почте;
- удалить вопрос.

Уведомление о каждом присланном вопросе автоматически отправляется модератору по электронной почте. Модератор либо пересылает вопрос эксперту, либо удаляет (например, в случае спама), а также присваивает вопросу какую-либо категорию. Если посетитель пометил свой вопрос как скрытый, но по каким-то причинам его необходимо опубликовать (например, указан некорректный e-mail и отправить ответ невозможно), то модератор задает дату снятия вопроса с публикации. Таким образом, отображаться на сайте будут вопросы, удовлетворяющие следующим условиям:

- есть ответ;
- либо вопрос не помечен как скрытый, либо дата снятия вопроса с публикации указана и больше текущей даты

Для каждого вопроса будем хранить следующие данные:

- id;
- имя автора;
- дата вопроса;
- собственно текст вопроса;
- город;
- e-mail автора;
- IP автора;
- id категории;
- отображать ли вопрос на сайте;
- дата снятия вопроса с публикации;
- отправлен ли вопрос эксперту;
- ответ на вопрос;
- отправлен ли ответ автору вопроса.

егистрация компонента в базе данных

Зарегистрируем наш компонент в базе данных, добавив запись в таблицу, содержащую данные о расширениях. Перейдите в **phpMyAdmin** (если вы работаете с **Denwer**, то для этого вам нужно ввести в адресной строке браузера http://localhost/tools/phpmyadmin), зайдите в базу данных, в которую вы установили Joomla, и откройте вкладку "**SQL**" (рис. 1.3).

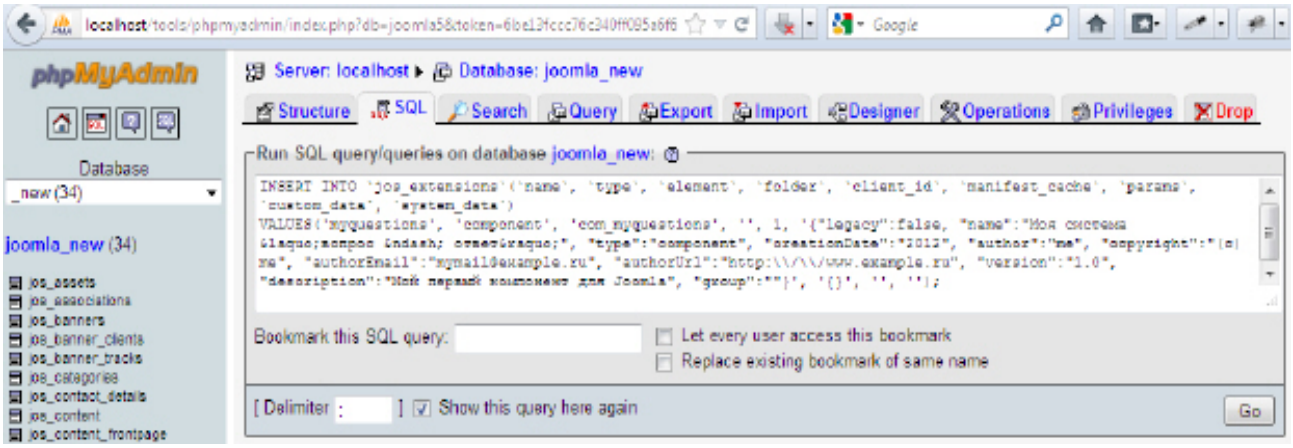


Рис. 1.3. Выполнение SQL-запроса

Теперь введите следующий SQL-запрос. Предполагается, что префикс таблиц вашей базы данных - "jos_". Если вы указали при установке Joomla другой префикс, замените "jos_" на него.

```
INSERT INTO `jos_extensions`(`name`, `type`, `element`,
  `folder`, `client_id`, `manifest_cache`, `params`, `custom_data`,
    `system_data`)
VALUES('myquestions', 'component', 'com_myquestions', '', 1,
'{"legacy":false, "name":"Моя система
&laquo;вопрос &ndash; ответ&raquo;", "type":"component",
"creationDate":"2012", "author":"me",
"copyright":"(c) me", "authorEmail":"mymail@example.ru",
"authorUrl":"http:\\/\\/www.example.ru",
 "version":"1.0",
"description":"Мой первый компонент для Joomla",
 "group":""}', '{}', '', '');
```

Как видите, мы задали название расширения - "myquestions", тип расширения - компонент и некоторую информацию о нем и его авторе.

Первые папки и файлы. Добавление пунктов меню

Создайте две папки под названием **com_myquestions**:

- в папке **/components**;
- в папке **/administrator/components**.

Создайте файл **/components/com_myquestions/myquestions.php** следующего содержания:

```
<?php
defined('_JEXEC') or die('Restricted access');
echo 'Моя система &laquo;вопрос &ndash; ответ&raquo;';
?>
```

Как видите, пока наш компонент просто будет выводить надпись "Моя система "вопрос - ответ"".

Не забудьте, что Joomla работает в кодировке UTF-8 и, следовательно, ваши PHP-файлы должны быть в той же кодировке. Например, в популярном текстовом редакторе Notepad++ кодировка изменяется с помощью пункта меню "**Кодировка**", из подпунктов которого нужно выбрать "**Преобразовать в UTF-8 без BOM**".

Теперь сохраните файл и обновите страницу http://localhost/joomla/index.php?option=com_myquestions в браузере. Страница примет следующий вид (рис. 1.4).

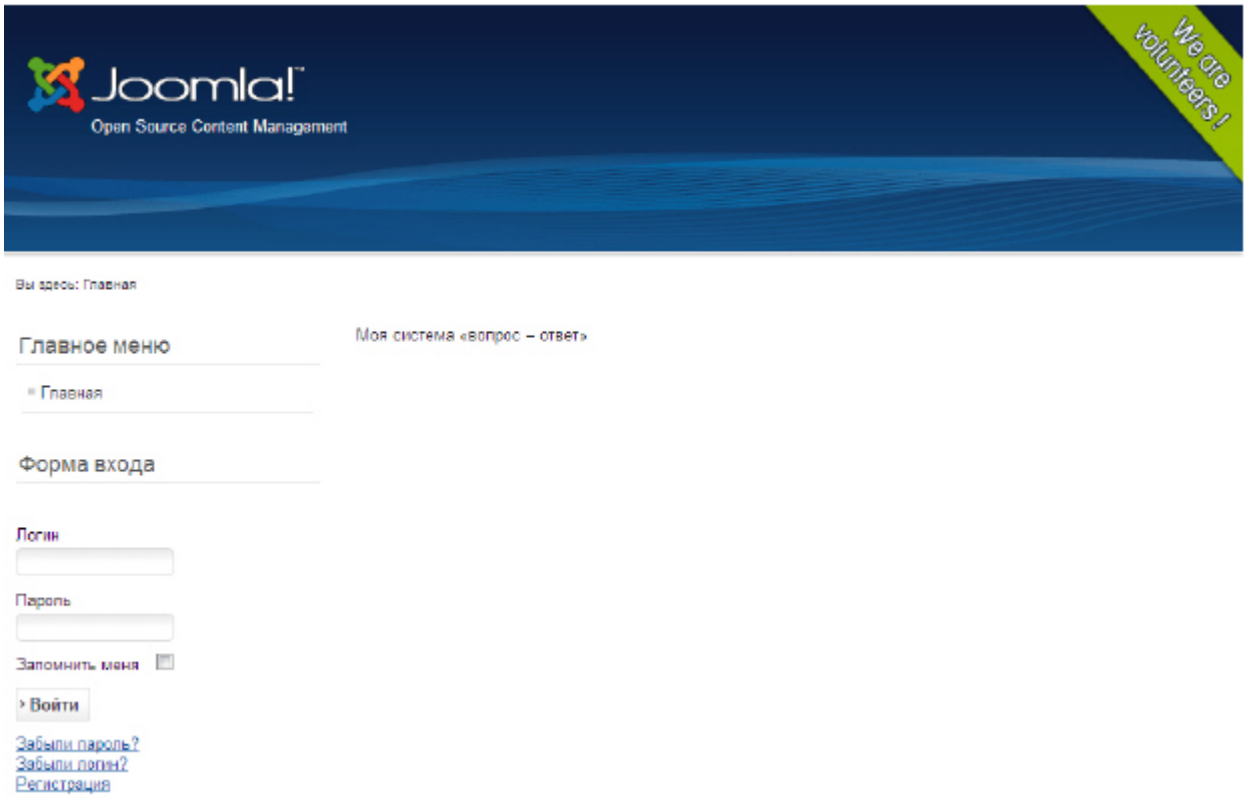


Рис. 1.4. Первый результат во фронтенде

Создайте файл **/administrator/components/com_myquestions/admin.myquestions.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
echo 'Моя система &laquo;вопрос &ndash; ответ&raquo;';
?>
```

Наберите в адресной строке браузера строку http://localhost/joomla/administrator/index.php?option=com_myquestions. Результат должен выглядеть так, как на рис. 1.5.

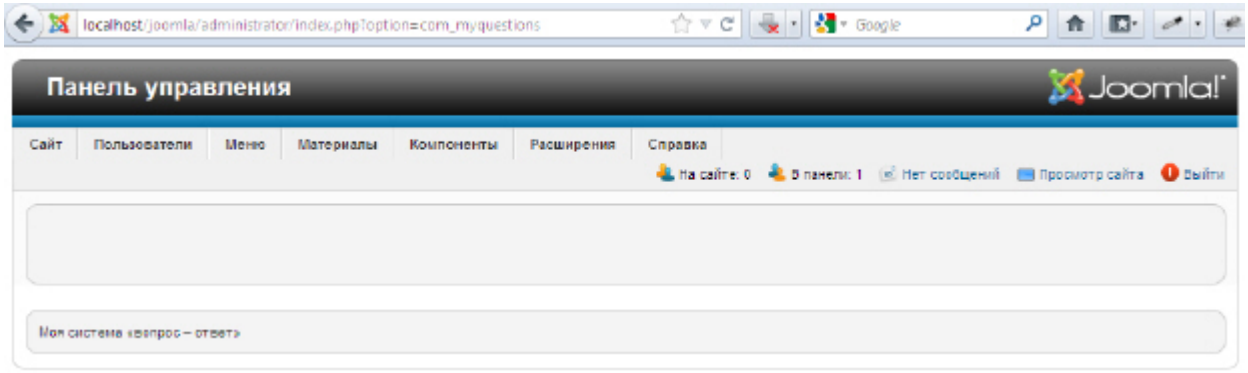


Рис. 1.5. Первый результат в бэкенде

Добавим эти ссылки в меню фронтенда и бэкенда. Для этого выясним, какой **id** был присвоен нашему компоненту в таблице **jos_extensions**. В **phpMyAdmin** войдите в эту таблицу и найдите расширение **com_myquestions**. Вероятно, оно находится в последней строке. Посмотрите, какое значение стоит в поле **extension_id**. Например, на рис. 1.6 видно, что в данном случае **id** равен 10006.

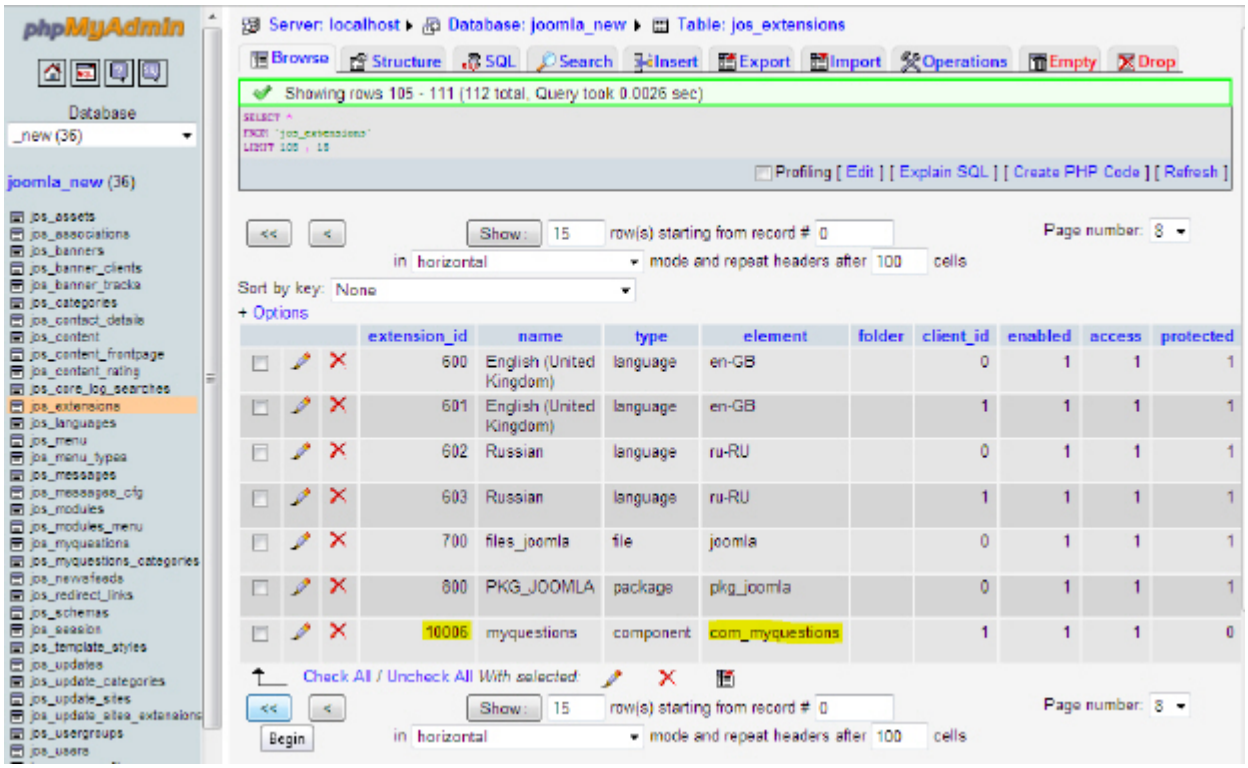


Рис. 1.6. Определение id компонента

Теперь выполните SQL-запрос, не забыв заменить "10006" на найденный вами id:

```
INSERT INTO `jos_menu`(`menutype`, `title`, `alias`, `path`, `link`, `type`,
  `level`, `component_id`, `access`, `img`,  `params`, `client_id`)
VALUES('menu', 'com_myquestions_menu', 'My Questions', 'My Questions',
'index.php?
option=com_myquestions', 'component',  1, 10006, 1, 'class:component', '', 1);
```

После выполнения данного запроса в меню "**Компоненты**" бэкенда появится новый подпункт со ссылкой на наш компонент. Однако он будет называться "**myquestions**", так как мы еще не задали перевод для строки com_myquestions_menu. Чтобы задать его, создайте языковой файл **/administrator/language/ru-RU/ru-RU.com_myquestions.sys.ini** следующего содержания:

```
COM_MYQUESTIONS_MENU="Моя система &laquo;вопрос &ndash; ответ&raquo;"
```

Зайдите в бэкенд. В меню "**Компоненты**" появился пункт "**Моя система "вопрос - ответ""** (рис. 1.7).

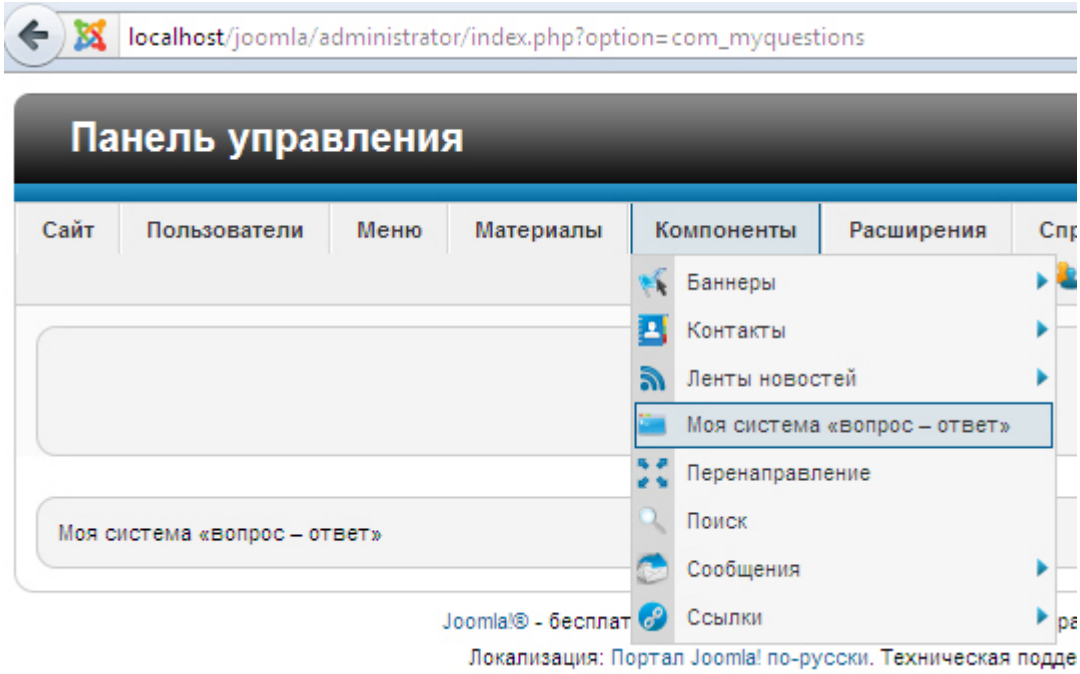


Рис. 1.7. Пункт меню в бэкенде

Теперь создайте пункт меню фронтенда. Для этого зайдите в "**Меню**" - "**Менеджер меню**", нажмите на ссылку "**Главное меню**", а затем на кнопку "**Создать**". Выберите тип пункта меню "**Внешний URL**", в поле "**Заголовок меню**" введите "**Моя система "вопрос - ответ""**, а в поле "**Ссылка**" - ссылку на фронтенд нашего компонента: http://localhost/joomla/index.php?option=com_myquestions и нажмите "**Сохранить**". Обновите любую страницу фронтенда и убедитесь, что появился новый пункт меню (рис. 1.8).

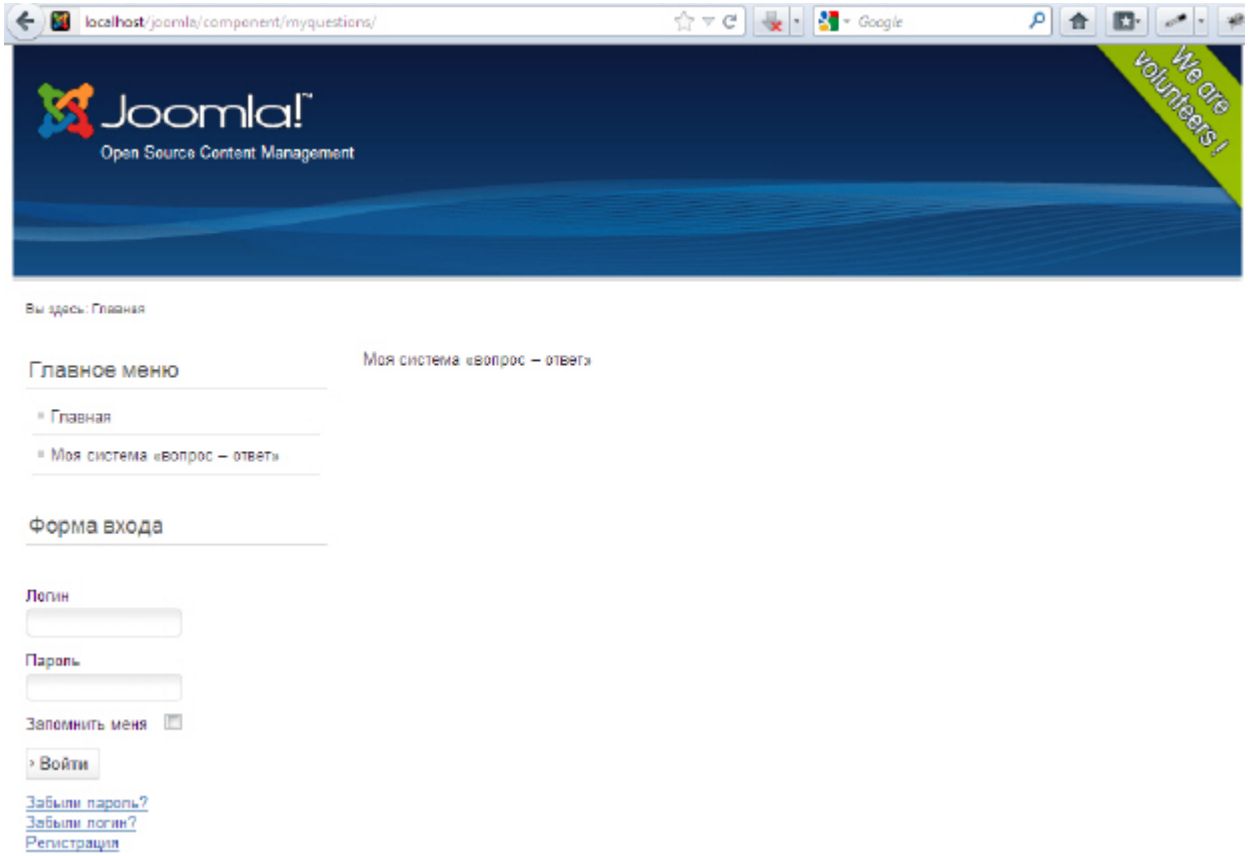


Рис. 1.8. Пункт меню во фронтенде

Создание панелей инструментов

Вспомните приведенное выше описание функционала, доступного администратору нашей системы. Соответственно, на панели инструментов для управления списком вопросов необходимы следующие кнопки:

- отправить уведомление о вопросе эксперту по электронной почте;
- присвоить вопросу категорию;
- отправить ответ автору вопроса по электронной почте;
- редактировать вопрос (в том числе: задать дату снятия вопроса с публикации; скрыть от посетителей сайта отдельные поля вопроса; ответить на вопрос);
- удалить вопрос.

Для начала создайте файл **/administrator/components/com_myquestions/toolbar.myquestions.html.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
class TOOLBAR_myquestions
{
  function _REPLY()
  {
    JToolBarHelper::title(JText::_('COM_MYQUESTIONS_TOOLBAR_TITLE'),
'generic.png');
    JToolBarHelper::custom('sendToExpert', 'send.png', '',
'COM_MYQUESTIONS_TOOLBAR_SEND_TO_EXPERT', false);
    JToolBarHelper::custom('sendAnswer', 'send.png', '',
'COM_MYQUESTIONS_TOOLBAR_SEND_ANSWER', false);
    JToolBarHelper::save();
```

```
    JToolBarHelper::apply();
    JToolBarHelper::cancel();
  }

  function _DEFAULT()
  {
    JToolBarHelper::title(JText::_('COM_MYQUESTIONS_TOOLBAR_TITLE'),
'generic.png');
    JToolBarHelper::editList('reply','COM_MYQUESTIONS_REPLY');
JToolBarHelper::deleteList(JText::_('COM_MYQUESTIONS_TOOLBAR_REMOVE_QUESTIONS_CO
NFIRMATION'));
  }
}
?>
```

Каждая функция класса TOOLBAR_myquestions соответствует отдельной панели инструментов. Как видите, мы задали две такие панели - первая из них будет отображаться над формой для ответа на вопрос, а вторая - над списком вопросов.

В коде, приведенном выше для ряда кнопок вместо стандартных подписей задаются собственные, которые необходимо перевести в языковом файле вместе с другими надписями. Поэтому создадим языковой файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini**:

```
COM_MYQUESTIONS_TOOLBAR_SEND_TO_EXPERT="Отправить эксперту"
COM_MYQUESTIONS_TOOLBAR_SEND_ANSWER="Отправить ответ"
COM_MYQUESTIONS_TOOLBAR_TITLE="Моя система &laquo;вопрос &ndash; ответ&raquo;"
COM_MYQUESTIONS_REPLY="Ответить / Редактировать"
COM_MYQUESTIONS_TOOLBAR_REMOVE_QUESTIONS_CONFIRMATION="Вы действительно хотите
удалить эти вопросы?"
```

Добавим код, который будет выбирать, какую из определенных нами панелей инструментов отображать. Создайте файл **/administrator/components/com_myquestions/toolbar.myquestions.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
require_once(JApplicationHelper::getPath('toolbar_html'));
switch($task)
{
  case 'reply':
    TOOLBAR_myquestions::_REPLY();
    break;
  default:
    TOOLBAR_myquestions::_DEFAULT();
    break;
}
?>
```

Вызов функции getPath() класса JApplicationHelper позволяет обратиться к файлу **toolbar.myquestions.html.php** без указания имени компонента, что удобно, если впоследствии понадобится изменить это имя.

Выражение switch используется для выбора одной из панелей инструментов в зависимости от значений переменной $task.

Обратите внимание, что код распределен по двум файлам - **toolbar.myquestions.php** и **toolbar.myquestions.html.php**, чтобы отделить логику обработки от непосредственного вывода информации.

Обновите страницу в бэкенде и выберите в меню "**Компоненты**" пункт "**Моя система "вопрос - ответ"**". Результат должен быть таким же, как на рис. 1.9.

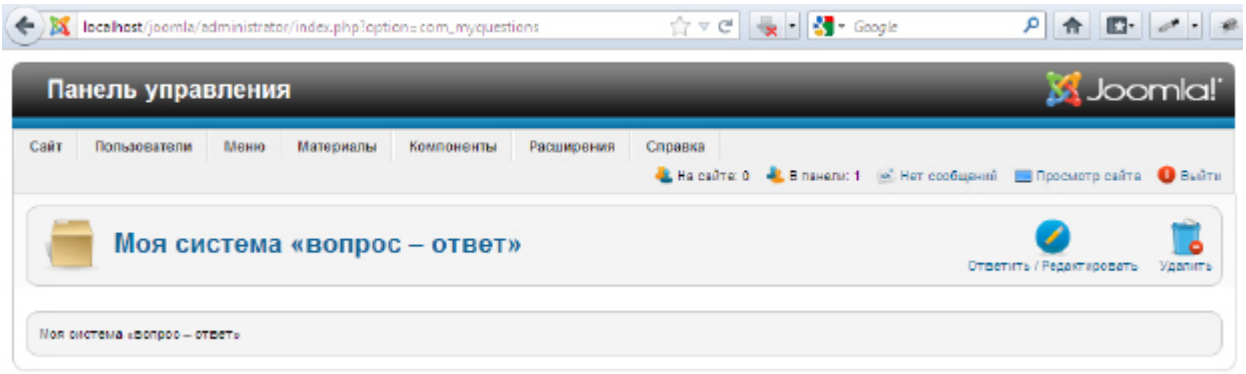


Рис. 1.9. Первая панель инструментов

Чтобы увидеть другую панель инструментов, добавьте строку &task=reply в конец URL: http://localhost/joomla/administrator/index.php?option=com_myquestions&task=reply (рис. 1.10).

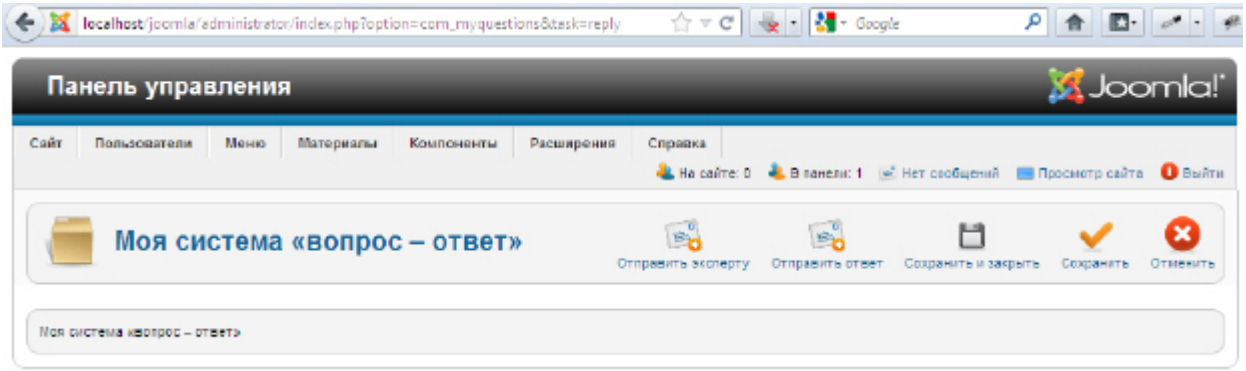


Рис. 1.10. Вторая панель инструментов

Каждая кнопка панели инструментов соответствует некоторой задаче. Когда пользователь нажимает на какую-либо кнопку, соответствующая задача добавляется к форме, и эта форма автоматически отправляется. Поскольку самих форм мы еще не создали, кнопки не работают.

Чтобы использовать для кнопки свою картинку, необходимо создать файл CSS, содержащий подобное выражение:

```
.icon-32-myiconname
{
     background-image: url(icon-32-myiconfile.png);
}
```

Класс JToolBarHelper будет искать для заданного значения iconname класс .icon-32- iconname. Название файла должно начинаться с "icon-32-".

В таком случае вызов функции JToolBarHelper::custom() будет выглядеть так:

```
JToolBarHelper::custom('someFunction', 'myiconname.png', '',
'Альтернативный текст', false, false);
```

Ваш файл CSS необходимо подключить в файле **admin.myquestions.php** следующим образом:

```
JHTML::stylesheet('delete.css', 'administrator/components/com_myquestions/');
```

Предполагается, что файлы **delete.css** и **icon-32-myiconfile.png** находятся в папке **/administrator/components/com_myquestions/**.

Ключевые термины

JAdministrator	- приложение, управляющее функциями для администрирования Joomla.
JApplication	- класс, позволяющий работать с очередью сообщений, осуществлять перенаправление браузера, получать параметры конфигурации сайта, определять тип запущенного приложения Joomla.
JFactory	- класс Joomla, реализующий паттерн "фабрика" и позволяющий получить доступ к глобальным объектам фреймворка.
JInstallation	- приложение, которое запускается при установке Joomla.
JRequest	- класс Joomla, использующийся для работы с переменными HTTP-запроса.
JSite	- приложение, отвечающее за компоновку и отображение фронтенда.
JToolBarHelper	- класс Joomla, содержащий методы, которые генерируют HTML-код для построения кнопок панелей инструментов.
XML-RPC	- приложение, позволяющее администрировать сайт Joomla удаленно.
Библиотека	- файл, который требуется для работы фреймворка или сторонних расширений.
Бэкенд	- система администрирования сайта.
Ключ	- эквивалент текста, подлежащего переводу.
Компонент	- основной тип расширений Joomla, вызов которого происходит при каждом обращении к Joomla.
Модуль	- расширение Joomla, использующееся для отображения небольших фрагментов контента, обычно в левой или правой колонке или верхней или нижней областях страницы.
Очередь сообщений	- массив строк, которые будут выведены на экран при следующей загрузке какой-либо страницы.
Перевод	- строка, содержащая перевод текста, соответствующего заданному ключу, на какой-либо язык.
Плагин	- расширение Joomla, позволяющее зарегистрировать функции и классы для обработки каких-либо событий, вызванных Joomla, например, поиск по сайту.
Приложение	- глобальный объект, использующийся для обработки запросов.
Уровень приложения	- часть архитектуры Joomla, которая состоит из приложений, расширяющих абстрактный класс JApplication.

Уровень расширений	- часть архитектуры Joomla, которая состоит из расширений фреймворка Joomla и приложений.
Уровень фреймворка	- часть архитектуры Joomla, которая обеспечивает ее базовую функциональность с помощью набора библиотек и плагинов и собственно ядра Joomla.
Фреймворк Joomla ("ядро")	- набор классов, обеспечивающих базовую функциональность Joomla (JDatabase, JUser, JForm, JEditor и т.д.).
Фронтенд	- часть сайта, доступная пользователю.
Шаблон	- расширение Joomla, отвечающее за внешний вид сайта.
Языковый файл	- расширение Joomla, позволяющее представить ее контент на нескольких языках.

Краткие итоги

Фреймворк Joomla состоит из трех уровней: уровень фреймворка, уровень приложения и уровень расширений.

Уровень фреймворка обеспечивает базовую функциональность Joomla с помощью набора библиотек и плагинов и собственно фреймворка Joomla.

Уровень приложения состоит из приложений, которые расширяют абстрактный класс **JApplication**.

Уровень расширений состоит из таких расширений фреймворка Joomla и приложений, как компоненты, модули, плагины и т.д.

Joomla делится на фронтенд - часть сайта, доступная пользователю, и бэкенд - систему администрирования сайта. Большинство компонентов для Joomla также делятся на фронтенд и бэкенд, и их код соответственно распределяется по двум папкам.

В Joomla определен ряд констант, хранящих значения путей, а также константа _JEXEC, позволяющая проверить, был ли скрипт вызван из Joomla, и DS - разделитель директорий.

Для создания мультиязыкового сайта используются языковые файлы, хранящие пары "ключ-значение", где ключ - это эквивалент какого-то текста, а значение - перевод этого текста на какой-либо язык.

Статический класс JFactory реализует паттерн "фабрика" и позволяет получить доступ к глобальным объектам фреймворка.

Вместо непосредственного использования глобальных массивов **$_GET, $_POST, $_REQUEST** и др. удобнее применять класс JRequest. Его методы пропускают данные, введенные пользователем, через фильтр во избежание инъекций.

В Joomla существует очередь сообщений - массив строк, которые будут выведены на экран при следующей загрузке какой-либо страницы. Добавляя сообщение в очередь, мы можем указывать его тип - сообщение, предупреждение или ошибка.

Глобальный объект JApplication позволяет работать с очередью сообщений, осуществлять перенаправление браузера, получать параметры конфигурации сайта, определять тип запущенного приложения Joomla.

Панели инструментов в бэкенде можно создавать с помощью класса JToolBarHelper, методы которого генерируют HTML-код для построения кнопок. Для отображения кнопок, которые часто используются в компонентах, существуют готовые методы этого класса. Можно также создать собственную кнопку.

Вопросы

1. Опишите структуру фреймворка Joomla.
2. Что такое фронтенд и бэкенд?
3. Какие константы предопределены в Joomla?
4. Каким образом реализована поддержка мультиязыковых сайтов?
5. Для чего используется класс JFactory?
6. В чем преимущество использования методов класса JRequest?
7. Что такое очередь сообщений?
8. Для чего используется глобальный объект JApplication?
9. Каким образом создаются панели инструментов в бэкенде?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. дополнительные материалы).

2. Лекция: Работа с базой данных

Подробно рассмотрена работа с базой данных средствами Joomla. Рассмотрены понятия реального и символического префиксов, этапы выполнения запроса к базе данных и класс JTable.

Цель лекции: Изучить, как происходит работа с базой данных средствами Joomla.

Префикс таблиц базы данных

Префикс таблиц базы данных - это строка, которая присоединяется к названию каждой таблицы Joomla в базе данных. Префикс задается при установке Joomla. В старых версиях по умолчанию использовался префикс "jos_", однако это создавало потенциальную уязвимость сайта, т.к. хакеры знали название таблицы с паролями пользователей - "jos_users". Теперь префикс, предлагаемый при установке, генерируется случайным образом.

Использование префикса позволяет разместить в одной базе данных несколько установок Joomla.

Различают реальный и символический префиксы. **Реальный префикс** - это то конкретное сочетание символов, которое используется в названиях таблиц базы данных.
Символический префикс - это сочетание "#__" (решетка и два знака подчеркивания), которое используется в запросах вместо реального префикса. При обработке запроса вместо символического префикса будет автоматически подставлен реальный. Например, при реальном префиксе "jos_" строка "#__mycomponent_mytable" превратится в "jos_mycomponent_mytable".

При разработке собственных расширений в SQL-запросах всегда указывается символический

префикс, а не реальный, так как в других установках Joomla почти наверняка будут использоваться другие реальные префиксы.

Выполнение запроса к базе данных

Чтобы выполнить запрос к базе данных Joomla, необходимо осуществить пять операций:

1. Получение ссылки на объект JDatabase.
2. Формирование запроса.
3. Задание запроса.
4. Выполнение запроса.
5. При необходимости - загрузка результата.

Получение ссылки на объект JDatabase

JDatabase - абстрактный класс, предоставляющий доступ к соединению с базой данных. Это соединение создается при инициализации приложения Joomla, а в коде своего расширения мы можем получить ссылку на него с помощью метода getDbo() статического класса JFactory:

```
$db =& JFactory::getDbo();
```

Формирование SQL-запроса

В старых версиях Joomla запросы формулировались в виде строки:

```
$query = 'SELECT * FROM #__categories';
```

В Joomla 1.6 появился объект JDatabaseQuery, методы которого позволяют упростить создание сложных SQL-запросов. Названия этих методов практически совпадают с ключевыми словами языка SQL: select(), from(), where(), having(), join() и т.д. Использование объекта JDatabaseQuery иллюстрирует следующий пример:

```
$db = JFactory::getDbo();
$query = $db->getQuery(true);
$query->select('id, name');
$query->from('#__users');
$query->order('name');
$query->where('username LIKE \'a%\'');
$db->setQuery($query);
echo $query->__toString();
```

В данном примере мы получаем из таблицы #__users отсортированный по алфавиту список id и имен пользователей, чьи логины начинаются на букву "a". Данный код выведет на экран следующий SQL-запрос:

```
SELECT id, name FROM #__users WHERE username LIKE 'a%' ORDER BY name
```

Как известно, употребляющиеся в запросе названия полей и таблиц рекомендуется заключать в ограничители, чтобы избежать совпадений с зарезервированными словами. Кроме того, строковые значения в запросах также берутся в кавычки. Методы nameQuote() и Quote() заключают, соответственно, названия и значения в правильные ограничители. Для MySQL

это обратные апострофы (``) для названий и обычные апострофы ('') для значений.

Рассмотрим пример использования этих методов:

```
$query = 'SELECT * FROM '.$db->nameQuote('#__users').' WHERE '.$db-
>nameQuote('username').'='.$db->Quote('admin');
```

Для базы данных MySQL с префиксом таблиц jos_ переменная $query примет следующее значение:

```
SELECT * FROM `jos_users` WHERE `username`='admin';
```

Задание запроса

Чтобы задать SQL-запрос для последующего выполнения, используется метод:

```
JDatabase setQuery(string $query, string $offset=0, string $limit=0)
```

где $query - это запрос, а $offset и $limit - соответственно смещение для начала выборки и количество выбираемых строк.

Например:

```
$db->setQuery($query, 0, 10);
```

Обратите внимание, что метод setQuery() не выполняет запрос, а только задает его.

Выполнение запроса

Без выборки данных

Для выполнения запроса, не требующего выборки данных (например, UPDATE или INSERT), используется метод mixed query().

При успешном выполнении запроса метод возвращает указатель на его результат, в противном случае - false. Например:

```
$db =& JFactory::getDBO();
$query = "UPDATE #__users SET block = 0 WHERE username LIKE 'a%'";
$db->setQuery($query);
$result = $db->query();
```

С выборкой данных

В классе JDatabase существуют методы для получения форматированного результата. Их можно разделить на следующие группы:

1. Получение одного значения: loadResult().
2. Получение одной строки таблицы: loadRow(), loadAssoc(), loadObject().
3. Получение одного столбца таблицы: loadResultArray().
4. Получение нескольких строк и нескольких столбцов: loadRowList(), loadAssocList(), loadObjectList().

Рассмотрим каждый из этих методов на примере таблицы #__categories, использующейся Joomla (таблица 2.1).

Таблица 2.1. Таблица #__categories

id	asset_id	parent_id	lft	rgt	level	path	extension	…	language
1	0	0	0	11	0		system	…	*
2	27	1	1	2	1	uncategorised	com_content	…	*
3	28	1	3	4	1	uncategorised	com_banners	…	*
4	29	1	5	6	1	uncategorised	com_contact	…	*
5	30	1	7	8	1	uncategorised	com_newsfeeds	…	*
6	31	1	9	10	1	uncategorised	com_weblinks	…	*

mixed loadResult()

Метод загружает значение первого столбца первой строки результирующей выборки. Используется для получения из базы данных какого-либо одного значения. При ошибке выполнения запроса метод вернет null, как и все рассмотренные ниже методы выборки данных.

Например, получим значение поля extension в записи под номером 2:

```
$db =& JFactory::getDbo();
$query = 'SELECT '.$db->nameQuote('extension').
      ' FROM '.$db->nameQuote('#__categories').
      ' WHERE '.$db->nameQuote('id').'='.$db->Quote('2');
$db->setQuery($query);
echo $db->loadResult();
```

Результатом выполнения данного запроса будет значение "com_content".

array loadRow()

Загружает первую строку результирующей выборки в виде массива. Если запрос возвращает больше одной строки, то метод вернет первую из них.

```
$db =& JFactory::getDbo();
$query = 'SELECT * FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
print_r($db->loadRow());
```

Результатом запроса будет следующий список (будем называть списком массив, индексами которого являются числа 0, 1, 2 и т.д.):

```
Array([0]=>1 [1]=>0 [2]=>0 [3]=>0 [4]=>11 [5]=>0 [6]=> [7]=>system [8]=>ROOT
 [9]=>root [10]=> [11]=> [12]=>1 [13]=>0 [14]=>0000-00-00 00:00:00
[15]=>1 [16]=>{} [17]=> [18]=> [19]=> [20]=>0 [21]=>2009-10-18 16:07:09
 [22]=>0 [23]=>0000-00-00 00:00:00 [24]=>0 [25]=>*)
```

array loadAssoc()

Метод загружает первую строку результирующей выборки в виде ассоциативного массива, ключами которого становятся названия полей таблицы. Если запрос возвращает больше одной строки, то метод вернет первую из них.

```
$db =& JFactory::getDbo();
$query = 'SELECT * FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
print_r($db->loadAssoc());
```

Результат запроса:

```
Array([id]=>1 [asset_id]=>0 [parent_id]=>0 [lft]=>0 [rgt]=>11 [level]=>0
[path]=> [extension]=
>system [title]=>ROOT [alias]=>root [note]=> [description]=>
[published]=>1 [checked_out]=>0 [checked_out_time]=>0000-00-00 00:00:00
[access]=
>1 [params]=>{} [metadesc]=> [metakey]=> [metadata]=> [created_user_id]=>0
[created_time]=>2009-10-18 16:07:09 [modified_user_id]=
>0 [modified_time]=>0000-00-00 00:00:00 [hits]=>0 [language]=>*)
```

object loadObject()

Метод загружает первую строку результирующей выборки в виде объекта класса stdClass, причем его полями становятся названия полей таблицы. Если запрос возвращает больше одной строки, то метод вернет первую из них.

```
$db =& JFactory::getDbo();
$query = 'SELECT * FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
print_r($db->loadObject());
```

Результат запроса:

```
stdClass Object([id]=>1 [asset_id]=>0 [parent_id]=>0 [lft]=>0 [rgt]=>11 [level]=
>0 [path]=> [extension]=>system [title]=>ROOT [alias]=>root [note]=>
[description]=> [published]=>1 [checked_out]=>0 [checked_out_time]=>0000-00-00
00:00:00 [access]=
>1 [params]=>{} [metadesc]=> [metakey]=> [metadata]=>
[created_user_id]=>0 [created_time]=>2009-10-18 16:07:09 [modified_user_id]=>0
[modified_time]=
>0000-00-00 00:00:00 [hits]=>0 [language]=>*)
```

array loadResultArray(int numinarray=0)

Метод загружает массив значений из результирующей выборки, полученных из одного столбца. Параметр numinarray используется для указания того, какой столбец нужно вернуть.

```
$db =& JFactory::getDbo();
$query = 'SELECT '.$db->nameQuote('extension').
      ' FROM '.$db->nameQuote('#__categories');
```

```
$db->setQuery($query);
print_r($db->loadResultArray());
```

Результатом будет следующий список:

```
Array([0]=>com_banners [1]=>com_contact [2]=>com_content
[3]=>com_newsfeeds [4]=>com_weblinks [5]=>system)
```

Данный метод позволяет перебирать в цикле столбцы таблицы:

```
$db =& JFactory::getDbo();
$query = 'SELECT '.$db->nameQuote('path').','.$db-
>nameQuote('extension').','.$db->nameQuote('language').
' FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
for ($i = 0; $i <= 2; $i++)
{
  $column = $db->loadResultArray($i);
  print_r($column);
  echo "<br>";
}
```

В результате на экран будет выведено:

```
Array([0]=> [1]=>uncategorised [2]=>uncategorised [3]=
>uncategorised [4]=>uncategorised [5]=>uncategorised)
Array([0]=>system [1]=>com_content [2]=>com_banners [3]=
>com_contact [4]=>com_newsfeeds [5]=>com_weblinks)
Array([0]=>* [1]=>* [2]=>* [3]=>* [4]=>* [5]=>*)
```

array loadRowList(int key)

Метод загружает список массивов или ассоциативный массив массивов. Если задан параметр key, то ключами возвращаемого массива будут значения поля, идущего в таблице под номером key, начиная с нуля.

```
$db =& JFactory::getDbo();
$query = 'SELECT * FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
print_r($db->loadRowList(7));
```

В данном примере из таблицы #__categories ядра Joomla извлекаются все записи, причем ключами полученного массива будут значения столбца №7, т.е. поля extension:

```
Array
(
  [system]=>Array([0]=>1   … [25]=>*)
  [com_content]=>Array([0]=>2   … [25]=>*)
  [com_banners]=>Array([0]=>3   … [25]=>*)
  [com_contact]=>Array([0]=>4   … [25]=>*)
  [com_newsfeeds]=>Array([0]=>5 … [25]=>*)
  [com_weblinks]=>Array([0]=>6 … [25]=>*)
)
```

Если не указать параметр key, то вместо ассоциативного массива мы получим список:

```
Array
(
  [0]=>Array([0]=>1 … [25]=>*)
  [1]=>Array([0]=>2 … [25]=>*)
  [2]=>Array([0]=>3 … [25]=>*)
  [3]=>Array([0]=>4 … [25]=>*)
  [4]=>Array([0]=>5 … [25]=>*)
  [5]=>Array([0]=>6 … [25]=>*)
)
```

array loadAssocList(string key='', string column='')

Метод загружает список ассоциативных массивов или ассоциативный массив ассоциативных массивов. Если задан параметр key, то ключами полученного массива будут значения столбца под названием key. Если задан параметр column, то в полученном массиве будет всего один столбец column.

```
$db =& JFactory::getDbo();
$query = 'SELECT * FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
print_r($db->loadAssocList('extension'));
```

Результат запроса:

```
Array
(
  [system]=>Array([id]=>1 [asset_id]=>0 ... [language]=>*)
  [com_content]=>Array([id]=>2 [asset_id]=>27 ... [language]=>*)
  [com_banners]=>Array([id]=>3 [asset_id]=>28 ... [language]=>*)
  [com_contact]=>Array([id]=>4 [asset_id]=>29 ... [language]=>*)
  [com_newsfeeds]=>Array([id]=>5 [asset_id]=>30 ... [language]=>*)
  [com_weblinks]=>Array([id]=>6 [asset_id]=>31 ... [language]=>*)
)
```

Как видим, ключи полученного массива - это значения поля extension, заданного параметром в метод loadAssocList().

Зададим значение второго параметра, чтобы получить только значение id для каждой строки таблицы:

```
$db =& JFactory::getDbo();
$query = 'SELECT * FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
print_r($db->loadAssocList('extension','id'));
```

Результат запроса:

```
Array([system]=>1 [com_content]=>2 [com_banners]=>3 [com_contact]=>4
[com_newsfeeds]=>5 [com_weblinks]=>6)
```

array loadObjectList(string key='')

Метод загружает список объектов stdClass или ассоциативный массив объектов stdClass. Если задан параметр key, то ключами полученного массива будут значения поля под названием key:

```
$db =& JFactory::getDbo();
$query = 'SELECT * FROM '.$db->nameQuote('#__categories');
$db->setQuery($query);
print_r($db->loadObjectList('extension'));
```

Результат запроса:

```
Array
(
  [system]=>stdClass Object([id]=>1 [asset_id]=>0 ... [language]=>*)
  [com_content]=>stdClass Object([id]=>2 [asset_id]=>27 ... [language]=>*)
  [com_banners]=>stdClass Object([id]=>3 [asset_id]=>28 ... [language]=>*)
  [com_contact]=>stdClass Object([id]=>4 [asset_id]=>29 ... [language]=>*)
  [com_newsfeeds]=>stdClass Object([id]=>5 [asset_id]=>30 ... [language]=>*)
  [com_weblinks]=>stdClass Object([id]=>6 [asset_id]=>31 .. [language]=>*)
)
```

Таблицы базы данных (класс JTable)

Класс JTable реализует паттерн Active Record и используется для управления таблицами базы данных.

Для каждой таблицы, которую вы будете создавать для своего компонента, необходимо создать класс, производный от JTable. Каждый такой класс помещается в отдельном файле в папке **/administrator/components/com_<имя компонента>/tables**. Имя класса строится по схеме Table<название таблицы>, а файла - **<название таблицы>.php**.

Для каждого поля таблицы необходимо создать одноименное поле класса.

Кроме того, создается конструктор класса, принимающий ссылку на объект JDatabase. Конструктор вызывает родительский конструктор, передавая ему название таблицы, название поля, являющегося первичным ключом таблицы, и объект JDatabase.

Например, пусть имеется таблица #__mycomponent_mytable из двух столбцов: id и name. Тогда производный от JTable класс должен выглядеть так:

```
class TableMytable extends JTable
{
  var $id = null;
  var $name = null;

  function __construct(&$db)
  {
    parent::__construct('#__mycomponent_mytable', 'id', $db);
  }
}
```

Чтобы использовать в коде вашего компонента файл, содержащий этот класс, нужно добавить папку tables в список директорий, в которых JTable может искать классы таблиц:

```
JTable::addIncludePath(JPATH_ADMINISTRATOR.DS.'components'.DS.'com_mycomponent'.
DS.'tables');
```

Теперь при создании экземпляра класса TableMytable Joomla будет искать файл **mytable.php**.

Для получения экземпляра данного класса используется метод getInstance():

```
object getInstance(string $type, string $prefix= 'JTable', array
$config=array())
```

где

$type	- вторая часть имени класса;
$prefix	- первая часть имени класса;
$config	- массив, содержащий настройки конфигурации.

В их числе может находиться объект-представитель базы данных, и тогда он будет использован вместо глобального объекта JDatabase. Например:

```
$row =& JTable::getInstance('mytable', 'Table');
```

Производный от JTable класс наследует в числе прочих методы bind(), store(), load() и delete(), позволяющие управлять записями таблицы без единой строки SQL-кода.

Создание/редактирование записи таблицы

Как правило, для создания или редактирования записи таблицы во фронтенде или бэкенде создается HTML-форма. Значения, введенные в нее пользователем, можно получить с помощью класса JRequest в виде массива. Полученный массив необходимо связать с объектом JTable. **Связывание** заключается в том, что каждому полю класса присваивается значение элемента массива, ключ которого совпадает с названием этого поля. Для этого используется метод

```
bool bind(mixed $src, mixed $ignore=array())
```

где:

$src	- ассоциативный массив или объект для связывания с экземпляром класса;
$ignore	- массив или разделенный пробелами список полей, которые нужно игнорировать при связывании.

Например:

```
if (!$row->bind(JRequest::get('post')))
  return JError::raiseWarning(500, $row->getError());
```

Обычно ошибка связывания возникает, если поля класса не соответствуют ключам массива, -

например, когда в HTML-форме в название элемента input вкралась опечатка.

Для сохранения записи используется метод bool store(bool $updateNulls=false)

Если параметр $updateNulls равен false, то те поля объекта JTable, которые имеют значение null, будут игнорироваться при связывании.

Пример использования метода store():

```
if (!$row->store())
  JError::raiseError(500, $row->getError());
```

Метод store() на основе хранящихся в $row значений генерирует запрос UPDATE или INSERT, в зависимости от значения id. Если запись создается впервые, то она не имеет значения id и будет сконструирован запрос INSERT, в противном случае - UPDATE. Это позволяет использовать данный метод как для создания новых записей, так и для обновления существующих.

Получение записи из таблицы

Для получения записи используется метод

```
bool load(mixed $keys = NULL, bool $reset = true)
```

где

$keys - первичный ключ записи, которую необходимо получить, или массив полей для поиска соответствий;

$reset - задает, будут ли перед получением новой записи заново присвоены полям значения по умолчанию.

Например:

```
$row->load($id);
```

Удаление записи

Для удаления записи используется метод

```
bool delete(mixed $id=null)
```

где $id - первичный ключ записи, которую требуется удалить. Если он не задан, то используется значение соответствующего поля объекта.

Пример:

```
$row->delete($id);
```

Управление полями ordering, checked_out/checked_out_time, published и hits

Существуют методы класса JTable для управления часто используемыми полями ordering, checked_out/checked_out_time, published и hits.

ordering

Поле ordering позволяет пользователю упорядочить список объектов по своему желанию. Чтобы пересчитать значения в поле ordering, используется метод

void reorder([string $where = ''])

При этом записи сортируются по значению ordering, а затем в это поле записываются натуральные числа, начиная с 1. Параметр $where позволяет задать условие ограничения выборки, к которой будет применена эта операция.

Пример использования метода:

```
$table->reorder();
```

Изменить значение ordering для одной записи, передвинув ее выше или ниже в списке, можно с помощью метода

```
void move(int $dirn, [string $where = ''])
```

где $dirn - величина, которая будет прибавлена к текущему значению ordering (отрицательное значение приведет к смещению записи вверх, а положительное - вниз).

Например, поднимем запись на одну строку вверх:

```
$table->load($id);
$table->move(-1);
```

checked_out/checked_out_time

Поля checked_out и checked_out_time используются для блокировки записей во избежание редактирования их несколькими пользователями одновременно. checked_out хранит id пользователя, работающего с записью в данный момент, а checked_out_time - время начала редактирования. Прежде чем заблокировать запись, необходимо проверить, не заблокирована ли она уже другим пользователем, с помощью метода

```
bool isCheckedOut(int $with=0, int $against=null)
```

где:

$with	- id пользователя, с которым нужно сравнить значение поля checked_out. Если запись заблокирована как раз этим пользователем, то функция вернет false, как и в том случае, если она не заблокирована вообще. В обоих этих случаях текущий пользователь имеет право работать с ней;
$against	- id пользователя, использующийся, если функция вызвана как статическая.

Для блокировки записей используется метод

```
bool checkOut(int $userId, mixed $pk=null)
```

где:

$userId	- id пользователя, блокирующего запись;
$pk	- первичный ключ записи, которую необходимо заблокировать. Если он не задан, используется значения соответствующего поля класса.

При этом в поле checked_out_time будет записано текущее время.

Для разблокировки записей используется метод bool checkIn(mixed $pk=null)

Рассмотрим пример использования этих методов:

```
$table->load($id);
$user =& JFactory::getUser();
if ($table->isCheckedOut($user->get('id')))
  die('Запись уже заблокирована другим пользователем');
echo 'Запись не заблокирована';
if (!$table->checkout($user->get('id')))
  die('Не удалось заблокировать запись с id текущего пользователя');
echo 'Заблокировали запись';
// работа с записью...
if (!$table->checkin($user->get('id')))
  die('Не удалось разблокировать запись');
echo 'Разблокировали запись';
```

published

Значение поля published показывает, опубликована ли запись. Чтобы изменить значение этого поля для одной или нескольких записей, используется метод

```
bool publish(mixed $pks=null, int $state=1, int $userId=0)
```

где:

$pks	- массив ключей записей, к которым необходимо применить операцию;
$state	- новое значение поля published (0 или 1);
$userId	- используется только когда в таблице существует также поле checked_out. При наличии в таблице этого поля метод publish() может быть применен только к тем записям, для которых checked_out равно 0 или заданному $userId.

Метод вернет true и в том случае, если какие-либо из записей были заблокированы и для них не удалось изменить значение published.

Пример использования этого метода:

```
$id_list = array($id);
```

```
$user =& JFactory::getUser();
if (!$table->publish($id_list, 1, $user->get('id')))
  die($table->getError());
```

hits

В поле hits хранится количество просмотров записи. Для увеличения этого значения на 1 используется метод

```
bool hit(mixed $pk=null)
```

где $pk - первичный ключ записи.

Например:

```
$table->hit();
```

Практика

Создание таблицы базы данных

Создайте таблицу для хранения вопросов, выполнив следующий SQL-запрос:

```
CREATE TABLE `jos_myquestions`
(
`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
`name` VARCHAR(255) NOT NULL,
`date` DATETIME NOT NULL,
`question` TEXT NOT NULL,
`city` VARCHAR(50) NULL,
`email` VARCHAR(50) NOT NULL,
`IP` VARCHAR(15) NOT NULL,
`id_cat` INT NOT NULL,
`published` TINYINT(1) NULL DEFAULT '1',
`expiration_date` DATETIME NULL DEFAULT '0000-00-00 00:00:00',
`senttoexpert` TINYINT(1) NULL DEFAULT '0',
`answer` TEXT NULL DEFAULT '',
`senttoauthor` TINYINT(1) NULL DEFAULT '0'
);
```

По умолчанию дата снятия вопроса с публикации имеет значение '0000-00-00 00:00:00'. Будем считать, что такая дата означает, что вопрос опубликован навсегда.

Для тестирования системы добавьте запись в таблицу, выполнив SQL-запрос:

```
INSERT INTO `jos_myquestions`(`id`, `name`, `date`, `question`, `city`, `email`,
`IP`, `id_cat`)
 VALUES(NULL, 'Аноним', '2012-01-01 09:00:00',
```

```
'Есть ли жизнь на Марсе', 'Москва', 'somebody@mail.ru', '12.345.67.890', '1')
```

Создание класса таблицы

Создайте в папке **/administrator/components/com_myquestions** папку **tables**. В этой папке создайте файл **question.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
class TableQuestion extends JTable
{
  var $id = null;
  var $name = null;
var $date = null;
  var $question = null;
  var $city = null;
  var $email = null;
  var $IP = null;
  var $id_cat = null;
  var $published = null;
  var $expiration_date = null;
  var $senttoexpert = null;
  var $answer = null;
  var $senttoauthor = null;

  function __construct(&$db)
  {
    parent::__construct('#__myquestions', 'id', $db);
  }
}
?>
```

Как видите, класс TableQuestion расширяет класс JTable. Каждому полю таблицы #__myquestions соответствует поле этого класса. Также перегружен конструктор __construct(), принимающий в качестве параметра объект-представитель базы данных и вызывающий родительский конструктор, используя название таблицы базы данных, первичный ключ и объект-представитель базы данных.

Создание формы для ответа на вопрос

Как и ранее, отделим HTML-вывод от логики обработки. PHP-код, необходимый для загрузки значений элементов формы, будет храниться в файле **admin.myquestions.php**, а код формы - в файле **admin.myquestions.html.php**. Откройте **admin.myquestions.php** и замените его содержимое следующим кодом:

```
<?php defined('_JEXEC') or die('Restricted access');
 $option = JRequest::getVar('option');
$task = JRequest::getVar('task');
require_once (JApplicationHelper::getPath('admin_html'));
JTable::addIncludePath(JPATH_COMPONENT.DS.'tables') ;
switch($task) {   case 'reply':     replyToQuestion($option);     break;
default:     break; }
function replyToQuestion($option)
{   $row =& JTable::getInstance('Question','Table');   $cid =
JRequest::getVar('cid', array(0), '',
```

```
 'array');   $id = $cid[0];   $row->load($id);
HTML_questions::replyToQuestion($row, $option); } ?>
```

Проверив, что код вызван из Joomla, мы используем выражение require_once(JApplicationHelper::getPath('admin_html')) для подключения файла **admin.myquestions.html.php**.

Затем с помощью JTable::addIncludePath() папка tables добавляется к списку директорий, в которых следует искать классы таблиц.

Переключатель switch() вызывает функцию, соответствующую значению переменной $task.

В функции replyToQuestion() создается экземпляр класса TableQuestion для управления записью таблицы. С помощью JRequest::getVar() из переменных запроса извлекается массив cid, хранящий идентификаторы записей. Так как эксперт будет отвечать только на один вопрос за раз, то мы выбираем первый идентификатор и загружаем соответствующую запись. Затем она передается в функцию вывода HTML_questions::replyToQuestion().

Теперь создайте файл **/administrator/components/com_myquestions/admin.myquestions.html.php**:

```
<?php
defined ('_JEXEC') or die ('Restricted access');
class HTML_questions
{
  function replyToQuestion($row, $option)
  {
    $editor =& JFactory::getEditor();
    ?>
    <form action = "index.php" method="post"
    name="adminForm" id="adminForm">
      <fieldset class="adminform">
        <table class="admintable" width=100%>
          <tr>
            <td width="100" class="key">
               <?php echo JText::_('COM_MYQUESTIONS_AUTHOR');?>:
            </td>
            <td>
              <input class="text_area" type="text" name="name" id="name"
              size="50" maxlength="255" value="<?php echo $row->name;?>"/>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_DATE');?>:
            </td>
            <td>
              <span class="text_area" type="text" name="date"
               id="date"><?php echo JHTML::_('date', $row-
>date,JText::_('DATE_FORMAT_LC3'));?></span>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_QUESTION');?>:
            </td>
            <td>
              <?php
```

```
                echo $editor->display('question',  $row->question, '100%',
'250', '40', '10');?>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_CITY');?>:
            </td>
            <td>
              <input class="text_area" type="text" name="city" id="city"
               size="50" maxlength="50" value="<?php echo $row->city;?>"/>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_EMAIL');?>:
            </td>
            <td>
              <input class="text_area" type="text" name="email" id="email"
               size="50" maxlength="50" value="<?php echo $row->email;?>"/>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_IP');?>:
            </td>
            <td>
              <span class="text_area" type="text" name="IP"
               id="IP"><?php echo $row->IP;?></span>
            </td>
          </tr>
          <tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_CATEGORY');?>:
            </td>
            <td>
              <input class="text_area" type="text" name="id_cat" id="id_cat"
              size="50" maxlength="250" value="<?php echo $row->id_cat;?>"/>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_PUBLISHED');?>:
            </td>
            <td valign="top">
              <?php
                if ($row->published == '1')
                  echo JText::_('JYES');
                else
                  echo JText::_('JNO');?>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_EXPIRATION_DATE');?>:
            </td>
            <td>
              <?php echo JHTML::_('calendar', $row->expiration_date,
'expiration_date', 'expiration_date', '%Y-%m-%d');?>
```

```
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_SENTTOEXPERT');?>:
            </td>
            <td valign="top">
              <?php
                if ($row->senttoexpert == '1')
                  echo JText::_('JYES');
                else
                  echo JText::_('JNO');?>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_ANSWER');?>:
            </td>
            <td>
              <?php
                echo $editor->display('answer',  $row->answer,'100%', '250',
'40', '10');?>
            </td>
          </tr>
          <tr>
            <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_SENTTOAUTHOR');?>:
            </td>
            <td valign="top">
                <?php
                if ($row->senttoauthor == '1')
                  echo JText::_('JYES');
                else
                  echo JText::_('JNO');?>
            </td>
          </tr>
        </table>
      </fieldset>
      <input type="hidden" name="id" value="<?php echo $row->id;?>"/>
      <input type="hidden" name="option" value="<?php echo $option;?>"/>
      <input type="hidden" name="task" value=""/>
    </form>
  <?php
  }
}
?>
```

Листинг .

Функция HTML_questions::replyToQuestion() выводит на экран уже заполненную форму, значения элементов которой берутся из объекта $row. Форме присвоено название adminForm, чтобы к ней можно было обращаться из JavaScript.

Классы JHTML и JEditor будут рассмотрены позже. Сейчас поясним только те выражения, в которых используются методы этих классов:

`echo JHTML::_('date', $row->date,` выводит дату $row->date в формате

`JText::_('DATE_FORMAT_LC3'));`	DATE_FORMAT_LC3 (один из стандартных форматов, заданных в Joomla).

`$editor = &JFactory::getEditor();`
`echo $editor->display('question',`
 	
`$row->question, '100%', '250', '40',`
`'10');`
	отображает выбранный администратором HTML-редактор. Если не выбран ни один редактор, то будет отображено поле <textarea>. В редакторе или поле <textarea> будет выведено значение $row->question.

`echo JHTML::_('calendar',`
 	
`$row->expiration_date,`
`'expiration_date', 'expiration_date',`
`'%Y-%m-%d');`
	выводит текстовое поле со значением $row->expiration_date и пиктограмму календаря, при нажатии на которую появляется календарь для выбора даты.

Перед закрывающим тегом </form> выводятся три скрытые элемента. Первый из них хранит значение id записи, т.к. оно необходимо для дальнейшего сохранения отредактированного вопроса. Элемент option хранит название текущего компонента для правильного редиректа в дальнейшем. Третьему скрытому элементу, task, не присвоено значения, чтобы JavaScript-код панели инструментов мог изменять его до отправки формы.

Осталось добавить перевод ключей COM_MYQUESTIONS_AUTHOR, COM_MYQUESTIONS_DATE и др. Откройте файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** и добавьте к его содержимому следующий код:

```
COM_MYQUESTIONS_AUTHOR="Автор"
COM_MYQUESTIONS_DATE="Дата вопроса"
COM_MYQUESTIONS_QUESTION="Текст вопроса"
COM_MYQUESTIONS_CITY="Город"
COM_MYQUESTIONS_EMAIL="e-mail"
COM_MYQUESTIONS_IP="IP-адрес"
COM_MYQUESTIONS_CATEGORY="Категория"
COM_MYQUESTIONS_PUBLISHED="Отображать ли вопрос на сайте"
COM_MYQUESTIONS_EXPIRATION_DATE="Дата снятия вопроса с публикации"
COM_MYQUESTIONS_SENTTOEXPERT="Отправлен ли вопрос эксперту"
COM_MYQUESTIONS_ANSWER="Ответ"
COM_MYQUESTIONS_SENTTOAUTHOR="Отправлен ли ответ автору вопроса"
```

Обратите внимание, что мы не задали перевод для слов "Да" и "Нет", а использовали ключи JYES и JNO, т.к. подобные распространенные слова уже переведены в файле **/administrator/language/ru-RU/ru-RU.ini**.

Наберите в адресной строке браузера ссылку http://localhost/joomla/administrator/index.php?option=com_myquestions&task=reply&cid[]=1. Должна появиться следующая страница (рис. 2.1).

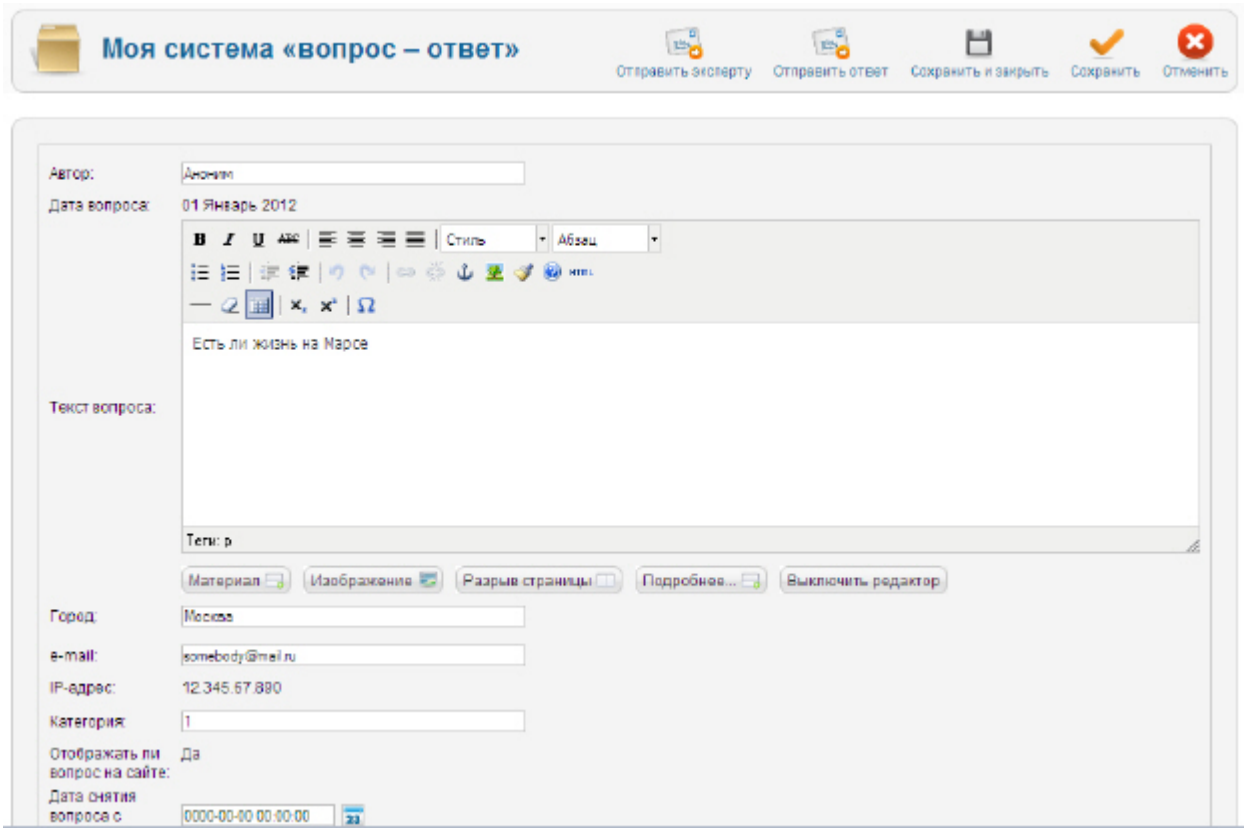


Рис. 2.1. Фрагмент формы для ответа на вопрос

Сохранение введенных данных

После того, как эксперт напечатал ответ на заданный вопрос и нажал кнопку "**Сохранить**", необходимо сохранить информацию в базе данных. Прежде всего, создайте две функции - save() и saveQuestion() - в файле **admin.myquestions.php**:

```
function save()
{
    $row =& JTable::getInstance('question', 'Table');
    if (!$row->bind(JRequest::get('post')))
    {
        echo "<script> alert('".$row->getError()."');
        window.history.go(-1); </script>\n";
        exit();
    }
    $row->question = JRequest::getVar('question', '', 'post', 'string',
JREQUEST_ALLOWRAW);
    $row->answer = JRequest::getVar('answer', '', 'post', 'string',
JREQUEST_ALLOWRAW);

    if (!$row->store())
    {
        echo "<script> alert('".$row->getError()."');
        window.history.go(-1); </script>\n";
        exit();
    }
    return $row;
}
function saveQuestion($option, $task)
{
    $row = save();
```

```
    global $app;
    if ($task == 'save')
        $app->redirect('index.php?option='.$option,
JText::_('COM_MYQUESTIONS_REPLY_SAVED'));
    else
        if ($task == 'apply')
            $app->redirect('index.php?option='.$option.'&task=
            reply&cid[]='.$row->id, JText::_('COM_MYQUESTIONS_REPLY_SAVED'));
}
```

Переменной $row присваивается значение экземпляра класса TableQuestion и вызывается функция bind() для связывания переменных, полученных из формы, с полями этого класса.

Для тех значений, которые вводились с помощью редактора Joomla, стандартный способ получения значений из массива JRequest::get('post') не подходит, т.к. функция bind() автоматически удаляет HTML-код, что приведет, в частности, к потере разрывов строк и тегов абзаца. Поэтому для получения значений question и answer в том виде, в котором они были введены в редакторе, используется функция getVar() класса JRequest. Данной функции передается имя переменной формы, значение по умолчанию, метод запроса, с помощью которого мы хотим получить данные (get/post), ожидаемый формат и флаг JREQUEST_ALLOWRAW, означающий, что данные не должны быть отфильтрованы.

Наконец, вызывается функция store() для сохранения вопроса в базе данных.

В функции saveQuestion() происходит вызов функции save(), а затем в зависимости от задачи, т.е. от того, какая кнопка была нажата, - "**Сохранить**" или "**Сохранить и закрыть**", - мы перенаправляем пользователя либо к той же странице редактирования вопроса, на которой он находится, но уже с сохраненными данными, либо к главной странице нашего компонента. В обоих случаях выводится сообщение о том, что данные были сохранены. Для перенаправления и вывода сообщения используется функция redirect() глобального объекта JApplication.

Добавьте задачу сохранения записи в переключатель switch() в файле **admin.myquestions.php** (выделенный код):

```
switch($task)
{
      case 'reply':
            replyToQuestion($option);
            break;
      case 'save':
      case 'apply':
            saveQuestion($option, $task);
            break;
      default:
            break;
}
```

Добавьте перевод для ключа COM_MYQUESTIONS_REPLY_SAVED в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini**:

```
COM_MYQUESTIONS_REPLY_SAVED="Данные сохранены"
```

Сохраните все ваши файлы и перейдите по ссылке http://localhost/joomla/administrator/index.php?option=com_myquestions&task=reply&cid[]=1. Напишите что-нибудь в поле для ответа и нажмите кнопку "**Сохранить и закрыть**". Вы должны увидеть следующую страницу (рис. 2.2).

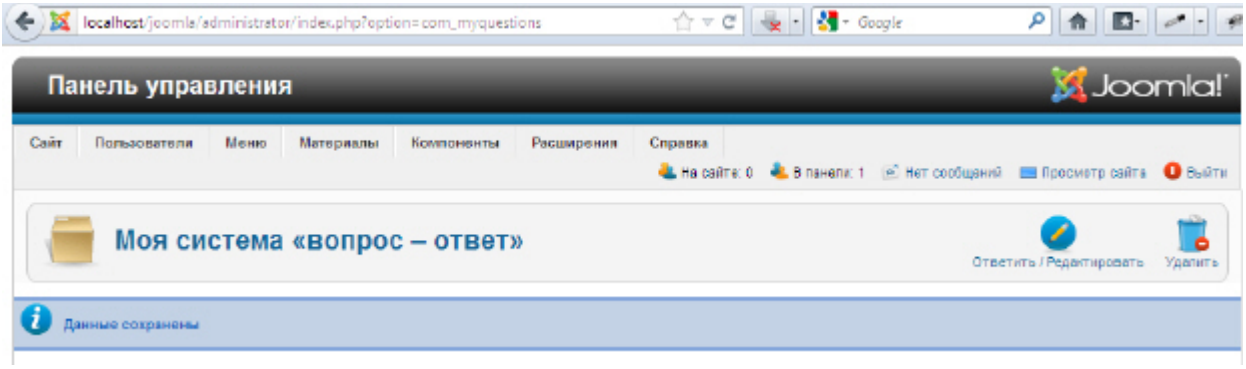


Рис. 2.2. Результат сохранения ответа на вопрос

С помощью phpMyAdmin вы можете проверить, что данные были сохранены в таблице базы данных jos_myquestions (рис. 2.3).

Рис. 2.3. Ответ на вопрос сохранен в базе данных

Вывод списка записей

Прежде всего, добавьте в файл **admin.myquestions.php** следующую функцию:

```
function showQuestions($option)
{
    $db =& JFactory::getDbo();
    $query = "SELECT * FROM #__myquestions";
    $db->setQuery($query);
    $rows = $db->loadObjectList();
    if ($db->getErrorNum())
    {
        echo $db->stderr();
        return false;
    }
    HTML_questions::showQuestions($option, $rows);
}
```

Эта функция загружает все записи из таблицы #__myquestions и передает их в виде массива $rows в следующую функцию, которую необходимо добавить в файл **admin.myquestions.html.php** в класс HTML_questions:

```
function showQuestions($option, &$rows)
  {
    $maxlen = 100;
  ?>
    <form action="index.php" method="post" name="adminForm">
      <table class="adminlist">
        <thead>
          <tr>
            <th width="20">
```

```
            <input type="checkbox" name="toggle" value=""
            onclick="checkAll(<?php echo count($rows);?>);"/>
          </th>
          <th class="title"><?php echo JText::_('COM_MYQUESTIONS_AUTHOR');?
></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_DATE');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_QUESTION');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_EMAIL');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_PUBLISHED');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_EXPIRATION_DATE');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_SENTTOEXPERT');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_ANSWER');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_SENTTOAUTHOR');?></th>
        </tr>
      </thead>
      <?php
        jimport('joomla.filter.output');
        $k = 0;
        for  ($i = 0,  $n = count($rows); $i < $n;  $i ++)
        {
          $row = &$rows[$i];
          $checked = JHTML::_('grid.id', $i, $row->id);
          $link = JFilterOutput::ampReplace('index.php?option=' .$option .
'&task=reply&cid[]='. $row->id);
      ?>
      <tr class="<?php echo "row$k";?>">
        <td><?=$checked?></td>
        <td><?=$row->name?></td>
        <td><?=JHTML::_('date', $row->date, JText::_('DATE_FORMAT_LC3'))?
></td>
        <td><?='<a href="'.$link.'">'.substr(strip_tags($row->question),
        0,$maxlen-1).'</a>'?></td>
        <td><?=$row->email?></td>
        <td align="center">
          <?php
            if ($row->published == '1')
              echo JText::_('JYES');
            else
              echo JText::_('JNO');?>
        </td>
        <td>
          <?php
            if ($row->expiration_date == '0000-00-00 00:00:00')
              echo JText::_('COM_MYQUESTIONS_DATE_NOT_DEFINED');
            else
              echo JHTML::_('date', $row->expiration_date,
JText::_('DATE_FORMAT_LC3'));?>
        </td>
        <td align="center">
          <?php
            if ($row->senttoexpert == '1')
              echo JText::_('JYES');
            else
              echo JText::_('JNO');?>
        </td>
        <td><?=substr(strip_tags($row->answer),0,$maxlen-1)?></td>
        <td align="center">
          <?php
            if ($row->senttoauthor == '1')
              echo JText::_('JYES');
```

```
            else
              echo JText::_('JNO');
          ?>
        </td>
      </tr>
      <?php
        $k = 1 - $k;
      }
      ?>
    </table>
    <input type="hidden" name="option"
    value="<?php echo $option;?>"/>
    <input type="hidden" name="task"
    value=""/>
    <input type= "hidden" name="boxchecked"
     value="0"/>
  </form>
<?php
}
```

Листинг .

Записи выводятся в таблице, для которой задан CSS-класс adminlist. Все заголовки таблицы, кроме первого, - это обычный текст. Первый заголовок является чекбоксом и используется для одновременного выделения всех отображенных записей.

Затем начинается цикл для вывода самих записей. Значение переменной $k меняется с 0 на 1 и обратно для того, чтобы переключаться между различными классами CSS для четных и нечетных строк, имеющими немного различающиеся свойства фона. С помощью вызова функции JHTML::_('grid.id') мы получаем HTML-код для чекбокса, который будет обрабатываться с помощью JavaScript.

Для каждого вопроса и ответа выводятся первые maxlen символов вместо его текста целиком. При этом с помощью функции strip_tags() отбрасываются теги, чтобы предотвратить ситуацию, когда граница обрезки текста может оказаться внутри тега.

Для перехода к форме ответа на вопрос для каждой записи выводится гиперссылка, которая пропускается через функцию JFilterOutput::ampReplace(), заменяющую амперсанды "&" на коды "&" в соответствии со спецификацией XHTML. Для подключения класса JFilterOutput в код вставлена строка jimport('joomla.filter.output').

Перед закрывающим тегом </form> расположены три скрытых элемента. Option и task были рассмотрены при анализе формы для ответа на вопрос. Значение boxchecked заключается в следующем. Когда пользователь ставит флажок в каком-либо из чекбоксов, значение boxchecked меняется на 1. Значение boxchecked, равное 0, возвращается, когда ни один из чекбоксов не отмечен. С помощью этого значения JavaScript обрабатывает список.

Для обработки случая, когда не выбрано никакой задачи, измените код переключателя switch в файле **admin.myquestions.php** следующим образом:

```
switch($task)
{
    case 'reply':
        replyToQuestion($option);
        break;
    case 'save':
    case 'apply':
        saveQuestion($option, $task);
```

```
        break;
    default:
        showQuestions($option);
        break;
}
```

Добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** строку:

```
COM_MYQUESTIONS_DATE_NOT_DEFINED="Дата не задана"
```

Теперь при загрузке http://localhost/joomla/administrator/index.php?option=com_myquestions должна появиться страница, как на рис. 2.4.

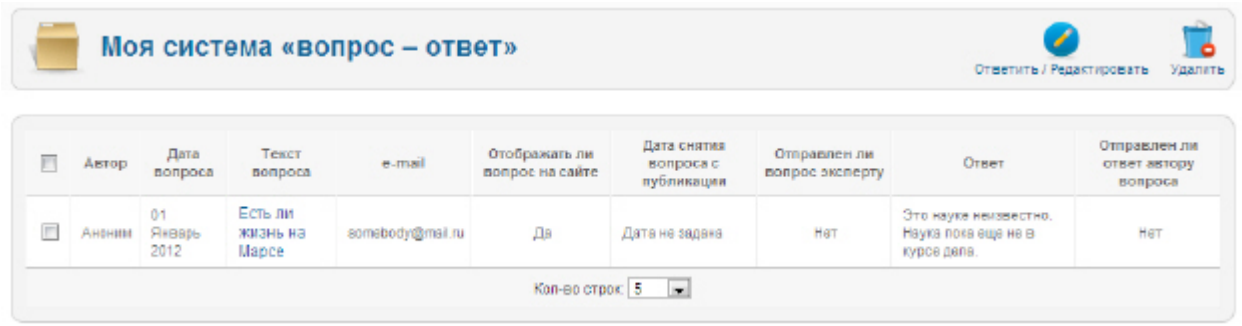


Рис. 2.4. Список вопросов

Удаление записей

Добавьте следующий оператор case в переключатель switch() в файле **admin.myquestions.php**:

case 'remove':

```
    removeQuestions($option);
    break;
```

Также добавьте функцию removeQuestions():

```
function removeQuestions($option)
{
    global $app;
    $cid = JRequest::getVar('cid', array(), '', 'array');
    $db =& JFactory::getDbo();
    if(count($cid))
    {
        $cids = implode(',', $cid);
        $query = "DELETE FROM #__myquestions WHERE id IN ($cids)";
        $db->setQuery($query);
        if (!$db->query())
        {
            echo "<script> alert('".$db->getErrorMsg()."');
            window.history.go(-1); </script>\n";
        }
    }
    $app->redirect('index.php?option=' . $option,
```

```
JText::_('COM_MYQUESTIONS_QUESTION_DELETED'));
}
```

Если в массиве cid есть элементы, то составляется строка из идентификаторов, разделенных запятыми, которая затем используется для построения запроса удаления соответствующих записей. В данном случае нельзя использовать метод JTable::delete(), т.к. он предназначен для удаления одной записи, а не нескольких.

Добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** строку:

```
COM_MYQUESTIONS_QUESTION_DELETED="Вопрос(ы) успешно удален(ы)"
```

Ключевые термины

JDatabase	- абстрактный класс, предоставляющий доступ к соединению с базой данных, создающемуся при инициализации приложения Joomla.
JDatabaseQuery	- класс, методы которого совпадают с ключевыми словами языка SQL и позволяют упростить создание сложных SQL-запросов.
JTable	- класс, реализующий паттерн Active Record и использующийся для управления таблицами базы данных.
Префикс таблиц базы данных	- строка, которая присоединяется к названию каждой таблицы Joomla в базе данных.
Реальный префикс	- то конкретное сочетание символов, которое используется в названиях таблиц базы данных.
Связывание	- процесс присвоения каждому полю производного от JTable класса значения элемента массива переменных запроса, так что ключ элемента совпадает с названием поля.
Символический префикс	- сочетание "#__" (решетка и два знака подчеркивания), которое используется в запросах вместо реального префикса.

Краткие итоги

При работе с базой данных различают реальный и символический префиксы. Реальный префикс используется в названиях таблиц базы данных, а символический префикс ("#__") используется в запросах вместо реального префикса. При обработке запроса вместо символического префикса будет автоматически подставлен реальный.

Чтобы выполнить запрос к базе данных Joomla, необходимо осуществить пять операций: получение ссылки на объект JDatabase (абстрактный класс, предоставляющий доступ к соединению с базой данных), формирование запроса, задание запроса, выполнение запроса, загрузка результата.

Запрос может быть сформулирован в виде строки либо разбит на составляющие и построен с помощью методов класса JDatabaseQuery.

Запрос задается для последующего выполнения методом setQuery(), а выполняется либо методом query(), либо, если нам необходимо получить результат, одним из методов для получения форматированного результата: loadResult(), loadRow() и т.д.

Для каждой таблицы, использующейся расширением, необходимо создать класс, производный от JTable. Для каждого поля таблицы необходимо создать одноименное поле этого класса. Производный от JTable класс наследует в числе прочих методы bind(), store(), load() и delete(), позволяющие управлять записями таблицы без единой строки SQL-кода. Когда компонент получает массив переменных запроса, он осуществляет связывание, то есть присваивает каждому полю этого класса значение элемента массива, ключ которого совпадает с названием данного поля.

Существуют методы класса JTable для управления часто используемыми полями ordering, checked_out/checked_out_time, published и hits.

Вопросы

1. Что такое реальный и символический префиксы?
2. Какие операции необходимо осуществить для выполнения запроса к базе данных Joomla?
3. Каким образом может быть сформулирован SQL-запрос?
4. Какие методы задают и выполняют запрос?
5. Для чего создается производный от JTable класс?
6. В чем заключается связывание?
7. Каким образом осуществляется управления часто используемыми полями?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. список вариантов в дополнительных материалах).

3. Лекция: Генерация элементов HTML (класс JHTML)

Лекция посвящена классу JHTML и поддерживающим классам. Рассмотрены методы для вывода элементов XHTML.

Цель лекции: Изучить готовые методы Joomla для генерации и отображения элементов XHTML и поведений JavaScript.

Основной метод класса JHTML

Joomla содержит методы для генерации и отображения элементов XHTML и поведений JavaScript. Эти методы вызываются с помощью

```
mixed _(string $type)
```

Несмотря на то, что в прототипе метода JHTML::_() указан всего один параметр, на практике их, как правило, бывает больше. Они интерпретируются так: по первому параметру метод _() определяет, какой метод необходимо вызвать, а остальные параметры передаются в этот метод.

Параметр $type может быть трех видов:

1. Название метода самого класса JHTML. Например, $type = 'image' приведет к вызову JHTML::image().
2. <имя файла>.<имя метода>. Будет вызван метод поддерживающего класса JHTML<Имя файла>.<имя метода>(). В имени поддерживающего класса первая буква <имени файла> станет заглавной. Например, если $type='select.genericlist', то будет вызван метод JHTMLSelect::genericlist().
3. <префикс>.<имя файла>.<имя метода> и будет вызван метод <префикс>.<Имя файла>.<имя метода>().

Методы класса JHTML

Класс JHTML содержит восемь методов для вывода элементов XHTML.

Поле для ввода даты и пиктограмма, по щелчку на которой появляется календарь

```
string calendar(string $value, string $name, string $id,
 string $format = '%Y-%m-%d', array $attribs = null)
```

где:

$value - значение даты;
$name - имя текстового поля;
$id - id текстового поля;
$format - формат даты;
$attribs - дополнительные атрибуты, которые должны быть выведены в теге <input>.

Для примера выведем текущую дату (рис. 3.1):

```
echo JHTML::_('calendar', date('Y-m-d',time()), 'created', 'created',
 '%Y-%m-%d', array('size'=>10,'style'=>"class='inputbox'"));
```

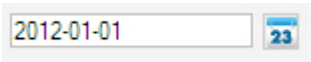


Рис. 3.1. Поле для ввода даты и пиктограмма для вывода календаря

Строка, содержащая дату в заданном формате и часовом поясе

```
string date(string $input = 'now', string $format = null, mixed $tz = true, bool $gregorian=false)
```

где:

$input	- строка в формате, подходящем для функции date();
$format	- формат, к которому необходимо привести дату;
$tz	- одна из временных зон, поддерживаемых PHP (их список можно найти на странице http://ww пользователя, при $tz=false - из настроек сервера;
$gregorian	- при false для форматирования даты будет использоваться локальный календарь.

Для примера выведем текущую дату:

```
echo JHTML::_('date',date('Y-m-d',time()), 'Y-m-d h:m',
'Europe/Moscow', false);
```

Элемент <iframe></iframe>

```
string iframe(string $url, string $name, array $attribs = null, string $noFrames = '')
```

где

$url	- относительный URL, будущее значение атрибута src;
$name	- название будущего элемента <iframe>;
$attribs	- ассоциативный массив атрибутов будущего тега <iframe>;
$noFrames	- содержимое будущего тега: текст, который будет показан, если браузер не поддерживает тег

Пример:

```
echo JHTML::_('iframe','index.php', 'myelement',
array('width'=>500,'height'=>300),
'Ваш браузер не поддерживает плавающие фреймы');
```

Элемент <img></img>

```
string image(string $file, string $alt, mixed $attribs = null,
bool $relative = false, bool $path_only = false)
```

где

$file - абсолютный или относительный URL изображения;
$alt - альтернативный текст;
$attribs - ассоциативный массив атрибутов будущего тега (может быть задан сразу в виде строки, нап
$relative - перебирать ли варианты пути к файлу;
$path_only - возвращать ли только путь к изображению или тег <img> со всеми атрибутами.

Для методов image(), script() и stylesheet() будет выполнена следующая процедура поиска файла. Если $file начинается с "http", то метод просто вернет $file. В противном случае он производит поиск файла, перебирая возможные варианты его названия в зависимости от браузера пользователя. Например, для значения "/pics/pic.png" и браузера Mozilla 5.0 будут перебираться варианты "/pics/pic.png", "/pics/pic_mozilla.png", "/pics/pic_mozilla_5.png", "/pics/pic_mozilla_5_0.png". При $relative=true перебираются также различные варианты пути к файлу в папке текущего шаблона и папке /media. В противном случае метод будет искать файл **JPATH_ROOT/<имя файла>** для каждого варианта названия файла.

Например, следующий код

```
echo JHTML::_('image','req.png','Восклицательный знак',
array('width'=>25,'height'=>25), true, false);
```

выведет

```
<img src="/joomla/templates/beez_20/images/req.png" alt="Восклицательный знак"
width="25" height="25"/>
```

Если последним параметром указать true, то эта же строка выведет на экран

```
/joomla/templates/beez_20/images/req.png
```

Обратите внимание, что этот путь мы не задавали, его обнаружил метод image().

Элемент <a>

```
string link(string $url, string $text, array $attribs = null)
```

где:

$text - текст ссылки;
$attribs - ассоциативный массив атрибутов будущего тега.

Пример:

```
echo JHTML::_('link','index.php','На главную страницу', array('title'=>'На
главную','id'=>'linktomainpage'));
```

Элемент <script></script>

```
mixed script(string $file, bool $framework = false, bool $relative = false,
 bool $path_only = false, bool $detect_browser = true)
```

где

$file	- путь к файлу;
$framework	- загружать ли фреймворк Javascript;
$path_only	- возвращать путь к файлу или добавить все найденные файлы к текущему документу;
$detect_browser	- определять ли браузер пользователя для включения файлов Javascript для этого браузер

При $path_only=false метод не возвращает никакого значения.

Пример:

```
echo JHTML::_('script', 'media/system/js/calendar.js', false, false, true,
false);
```

Элемент <link rel="stylesheet" style="text/css"/>

```
mixed stylesheet(string $file, array $attribs = array(), bool $relative = false,
 bool $path_only = false, bool $detect_browser = true)
```

Все параметры аналогичны рассмотренным выше.

Пример:

```
echo JHTML::_('stylesheet', 'media/system/css/system.css',
array("title"=>"Название стиля","media"=>"screen, print"), false, true, false);
```

Всплывающая подсказка

```
string tooltip(string $tooltip, mixed $title = '', string $image =
'tooltip.png', string $text = '', string $href = '',
string $alt = 'Tooltip', string $class = 'hasTip')
```

где

$tooltip	- текст подсказки;
$title	- название подсказки или ассоциативный массив с ключами title, image, text, href, alt и соответствующими значениями;
$image	- изображение, для которого будет выводиться подсказка (если не задано значение $text);
$text	- текст, для которого будет выводиться подсказка (если не задано значение $image);
$href	- URL, на который будет произведен переход по щелчку на изображение или текст;

$alt - значение атрибута alt тега <img>;
$class - название класса элемента , в который будет заключен тег <img>.

Изображение с подсказкой:

```
echo JHTML::tooltip('Текст подсказки', 'Заголовок подсказки', 'tooltip.png', '',
'http://www.mysite.ru');
```

Текст с подсказкой:

```
echo JHTML::tooltip('Текст подсказки', 'Заголовок подсказки', '',
  'Наведите на этот текст курсор мыши, чтобы увидеть подсказку');
```

Поддерживающие классы

Рассмотрим некоторые методы поддерживающих классов. Полный список этих методов можно найти в документации, однако на текущий момент она не достаточно полная и содержит ошибки, поэтому для детальной справки по какому-либо методу вы можете обратиться напрямую к исходным кодам в папке **/libraries/joomla/html/html**.

JHTMLBehavior

Методы этого класса загружают код Javascript в заголовок документа. Методы JHTMLBehavior позволяют вывести календарь, дерево элементов, файловый загрузчик и некоторые другие элементы управления. Рассмотрим один из методов, позволяющий вывести всплывающее модальное окно без перезагрузки страницы:

```
void modal(string $selector = 'a.modal', array $params = array())
```

где

$selector - селектор класса;
$params - массив параметров, ключи которого могут быть следующими: ajaxOptions, size, shadow, onOpe

В следующем примере выводятся две ссылки. При нажатии на первую из них появляется всплывающее окно с изображением, на вторую - с веб-страницей.

```
<?php
  JHTML::_('behavior.modal');
?>
<a href="media/system/images/notice-info.png"
class="modal" rel="{size: {x: 100, y: 100},
handler:'iframe'}" >Щелкните, чтобы увидеть изображение</a><br/>
<a href="http://www.mysite.ru" class="modal"
rel="{size: {x: 700, y: 500}, handler:'iframe'}" >Щелкните, чтобы открылось окно
с веб-страницей</a>
```

Первое окно выглядит так, как на рис. 3.2.

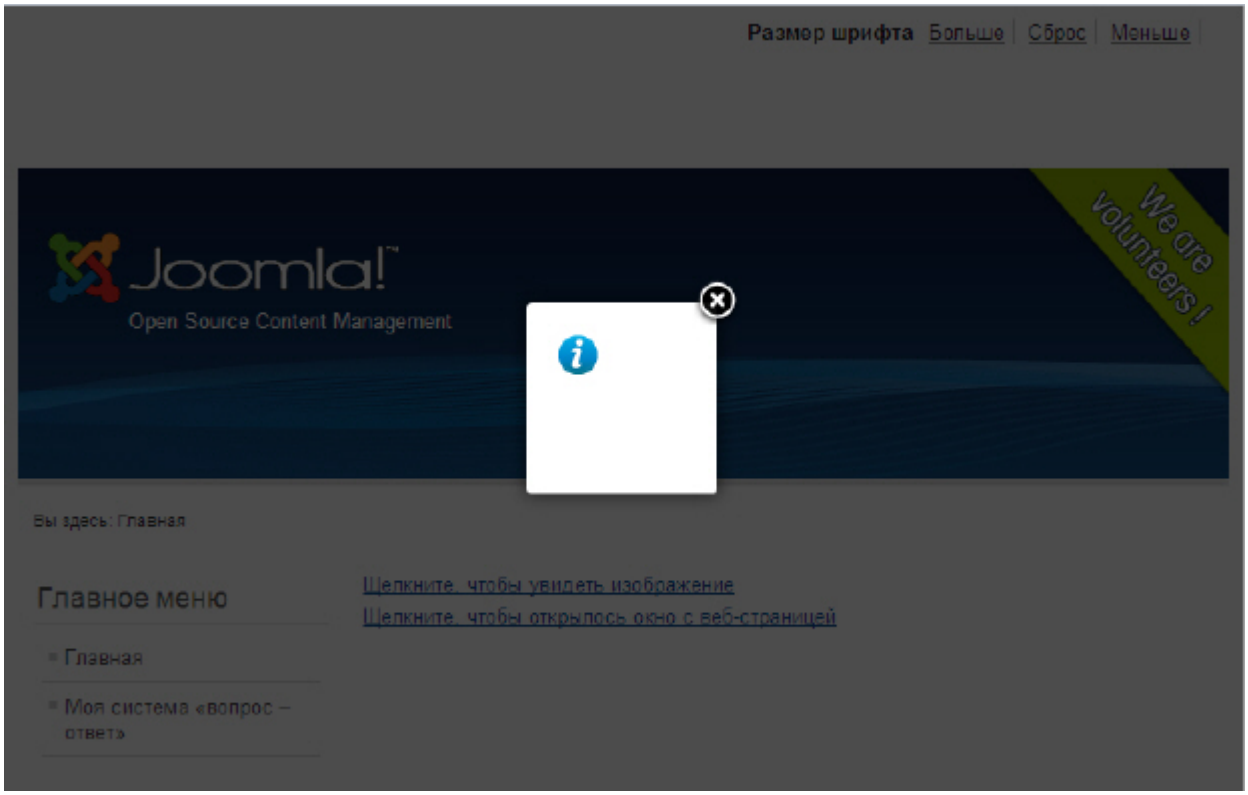


Рис. 3.2. Модальное окно

JHTMLEmail

Содержит один метод для скрытия адреса электронной почты в целях его защиты от спам-ботов:

```
string cloak(string $mail, bool $mailto=1, string $text='', bool $email=1)
```

где

$mail - e-mail;
$mailto - выводить ли e-mail в виде ссылки … или в виде текста;
$text - текст этой ссылки;
$email - является ли $text адресом электронной почты.

Пример:

```
echo JHtml::_('email.cloak', 'admin@mysite.ru', 1, 'Написать администратору',
0);
```

JHTMLForm

Содержит один метод, который возвращает код скрытого поля формы для уменьшения риска CSRF-атак: string token()

Именем получившегося элемента формы станет сгенерированная случайным образом строка, которая используется для проверки того, что запрос был сделан из конкретных формы и сессии.

Для предотвращения CSRF-атак вставляйте в каждую форму своего компонента строку:

```
<?php echo JHTML::_('form.token');?>
```

JHTMLGrid

Методы данного класса используются для вывода в таблице в панели управления таких элементов, как чекбокс, пиктограмма для переключения состояния "**опубликовано**"/"**не опубликовано**", для отображения заголовка столбца как ссылки для сортировки по этому столбцу и др.

Таблица должна располагаться внутри формы под названием adminForm, обязательно включающей два скрытых поля: boxchecked со значением по умолчанию 0 и task.

Для создания чекбокса используется метод

```
mixed id(int $rowNum, int $recId, bool $checkedOut=false, string $name='cid')
```

где

$rowNum	- номер строки в таблице;
$recId	- id записи;
$checkedOut	- отмечен ли элемент;
$name	- имя элемента формы.

Метод возвращает html-код чекбокса, если элемент не отмечен, и пустую строку в противном случае.

Для создания пиктограммы, по щелчку на которой можно изменять значение поля published с 0 на 1 и обратно, используется

```
string published(mixed $value, int $i, string $img1 = 'tick.png', string $img0 =
'publish_x.png', string $prefix='')
```

где

$value	- объект или только значение поля published;
$i	- номер строки в таблице;
$img1	- изображение при published = 1;
$img0	- изображение при published = 0;
$prefix	- префикс, который будет добавлен к названиям по умолчанию задач publish/unpublish.

Для примера выведем элементы массива $rows как строки таблицы, добавив для каждой

записи ячейки с чекбоксом и значком **"опубликовано"/"не опубликовано"**:

```
$k = 0;
for ($i = 0, $n = count($rows); $i < $n;  $i ++)
{
  $row = &$rows[$i];
  ?>
  <tr class="<?="row$k"?>">
    <td><?=JHTML::_('grid.id', $i, $row->id)?></td>
    <td><?=$row->name?></td>
    <td align="center"><?=JHTML::_('grid.published', $row, $i)?></td>
  </tr>
  <?php
  $k = 1 - $k;
}
```

Пример результата приведен на рис. 3.3.

Рис. 3.3. Чекбокс и пиктограмма "опубликовано/не опубликовано"

Обратите внимание, что методы класса JHTMLGrid лишь отображают какой-либо элемент, но не добавляют код для его обработки. В приведенном выше примере кнопки **"опубликовано"/"не опубликовано"** отображаются, но не работают. Чтобы это исправить, необходимо зарегистрировать задачи publish и unpublish (если вы не изменили их названия, задав какой-либо префикс) и написать функцию для их обработки, которая будет вызывать метод JTable::publish().

JHTMLImage

Содержит два метода для поиска изображения в фронтенде и бэкенде соответственно. Каждый из них ищет либо изображение из директории images текущего шаблона сайта/панели управления, либо, если файла с заданным именем там нет, изображение из заданной директории.

```
string site(string $file, string $folder = '/images/system/', int $altFile =
null, string $altFolder = '/images/system/',
string $alt = null, array $attribs = null, bool $asTag = true)
string administrator(string $file, string $folder = '/images/', int $altFile =
null, string $altFolder = '/images/',
string $alt = null, array $attribs = null, bool $asTag = true)
```

где

$file	- имя файла;
$folder	- путь к файлу. Будет использован, если в директории images текущего шаблона не нашлось файла с заданным именем;
$altFile	- если не задано, то используются значения $file и $folder, при $altFile = -1 метод возвращает пустую строку, при других значениях используются $altFile и

$altFolder;
$altFolder - другой путь к файлу;
$alt - будущее значение атрибута alt;
$attribs - ассоциативный массив атрибутов;
$asTag - вернуть весь тег <img> с его содержимым или только путь.

Пример:

```
echo JHTML::_('image.site', 'notice-info.png', '/media/system/images/');
```

JHTMLList

Методы класса JHTMLList используются для создания списков некоторых значений.

Список для выбора одного из существующих в Joomla уровней доступа

```
string accesslevel(object &$row)
```

где $row - объект, имеющий поле access.

Пример:

```
$query = 'SELECT id,access FROM #__content WHERE id = 1';
$db =& JFactory::getDBO();
$db->setQuery($query);
$article = $db->loadObject();
echo JHTML::_('list.accesslevel', $article);
```

Результат показан на рис. 3.4.

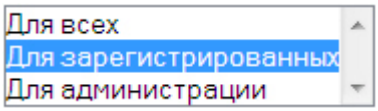


Рис. 3.4. Список для выбора уровня доступа

Список для выбора изображения

```
array images(string $name, string $active = NULL, string $javascript =
NULL,
string $directory = NULL, string $extensions =  "bmp|gif|jpg|png")
```

где

$name - имя поля;
$active - выбранный по умолчанию элемент;
$javascript - дополнительный код Javascript, который будет выведен внутри тега <select>;
$directory - директория, в которой хранятся изображения. Например: **components/com_mycomponent/images**. Если значение не задано, будет

использоваться директория images;
$extensions - список допустимых расширений

Например, вывод на экран JHTML::_('list.images','imglist') приведет к отображению такого списка (рис. 3.5).

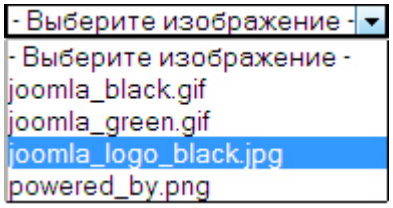


Рис. 3.5. Список для выбора изображения

Список незаблокированных пользователей

```
string users(string $name, string $active, int $nouser = 0, string $javascript =
NULL, string $order = 'name', string $reg = 1)
```

где

$name - название элемента <select>;
$active - выбранный по умолчанию пользователь;
$nouser - добавлять ли пункт, означающий отсутствие пользователя;
$javascript - дополнительный код Javascript, который будет выведен внутри тега <select>;
$order - имя поля для сортировки списка;
$reg - исключить пользователей из группы "**Зарегистрированные**".

Например, вывод на экран JHTML::_('list.users','usrlist','42', 1, NULL, 'id', $reg = 1) приведет к отображению такого списка (рис. 3.6).

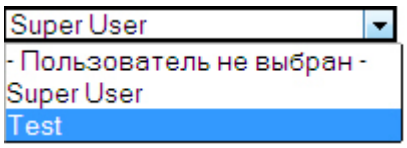


Рис. 3.6. Список для выбора пользователя

Список категорий

```
string category(string $name, string $extension, string $selected = NULL, string
$javascript = NULL,
string $order = null, int $size = 1, bool $sel_cat = 1)
```

где

$name - название элемента <select>;
$extension - расширение, к которому относятся категории;
$selected - выбранное по умолчанию значение;
$javascript - дополнительный код Javascript, который будет выведен внутри тега <select>;
$size - высота списка (значение атрибута size тега <select>);
$sel_cat - при null строки "**Выберите категорию**" в списке не будет.

Пример:

```
echo JHTML::_('list.category', 'catlist', 'com_content', 2,
'onclick="someFunc()"', 'id', 1, 1);
```

Данный код выведет на экран список, представленный на рис. 3.7.

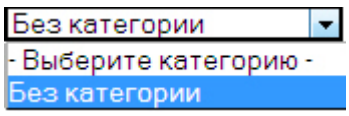


Рис. 3.7. Список для выбора категории

JHTMLMenu

Класс используется для отображения элементов меню и не представляет интереса для использования в расширениях.

JHTMLSelect

Класс JHTMLSelect используется для генерации кода списков.

Выпадающий список

```
string genericlist(array $data, string $name, mixed $attribs = null, string
$optKey = 'value',
string $optText = 'text', mixed $selected = null, mixed $idtag = false, bool
$translate = false)
```

где

$data - массив данных для отображения. Каждый элемент может быть сгенерирован с помощью метода JHTMLSelect.option() или другим образом;
$name - имя элемента HTML;
$attribs - дополнительные атрибуты тега <select>. Могут быть заданы как массив атрибутов или массив опций с ключами list.attr, id, list.select и др.;
$optKey - из какого поля объекта, представляющего собой элемент массива $data, брать значение элементов <option>. При $optkey = null будут использованы ключи массива;

$optText	- из какого поля объекта, представляющего собой элемент массива $data, брать текст элементов <option>. Тег <option> формируется так: <option value="$optkey"> $optText</option>;
$selected	- значение выбранного по умолчанию элемента;
$idtag	- id тега <select>;
$translate	- пропускать ли текст через функцию JText::_().

Пример:

```
$query = 'SELECT id,title FROM #__content';
$db =& JFactory::getDBO();
$db->setQuery($query);
$content = $db->loadObjectList();
echo JHTML::_('select.genericlist', $content, 'genlist',
'size='.count($content), 'id', 'title', 2);
```

Так как мы передали в функцию genericlist() $selected=2, то в сгенерированном списке по умолчанию выделен элемент с id, равным 2 (рис. 3.8).

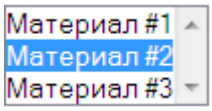


Рис. 3.8. Список для выбора материала

Объект, представляющий элемент <option> выпадающего списка

```
object option(string $value, string $text = '', mixed $optKey = 'value', string
$optText = 'text', bool $disable = false)
```

где

$value	- значение элемента <option>;
$text	- текст элемента;
$optKey	- если это строка, то имя того поля возвращаемого объекта, в котором будет храниться значение option.label, option.text;
$optText	- имя того поля возвращаемого объекта, в котором будет храниться текст элемента <option>;
$disable	- добавлять ли к тегу атрибут disable.

Рассмотрим пример:

```
$query = 'SELECT id,title FROM #__content';
$db =& JFactory::getDBO();
$db->setQuery($query);
$content = $db->loadObjectList();
foreach ($content as $c)
  $data[] = JHTML::_('select.option', $c->id, $c->title, 'my_value', 'my_text');
echo JHTML::_('select.genericlist', $data, 'genlist', 'size='.count($data),
'my_value', 'my_text', 2);
```

В данном примере для каждого материала сайта с помощью метода JHTMLSelect.option() создается объект, представляющий элемент <option>:

```
JObject Object ([_errors:protected]=>Array () [my_value]=>1 [my_text]=>Материал #1 [disable]=>)
```

Таким образом, значения $optKey и $optText, переданные в функцию option(), стали названиями полей получившегося объекта, а значения $c->id и $c->title - значениями этих полей. Теперь в функцию JHTMLSelect.genericlist() необходимо передать те же названия полей, что и в option(). Если не передать их, то функция genericlist() будет по умолчанию искать в объектах массива $data поля $value и $text, которых там нет.

Результат работы данного примера будет выглядеть так же, как и результат предыдущего.

Список целых чисел

```
string integerlist(int $start, int $end, int $inc, string $name, mixed $attribs = null, mixed $selected = null, string $format = '')
```

где

$start	- первое число последовательности;
$end	- последнее число последовательности;
$inc	- шаг;
$name	- имя тега;
$attribs	- массив атрибутов тега;
$selected	- значение выбранного по умолчанию элемента;
$format	- формат вывода числа для функции printf().

Пример:

```
echo JHTML::_('select.integerlist', 1, 10, 1, 'intlist', 'size=10', 3, '%02d');
```

Этот код выведет список, представленный на рис. 3.9.

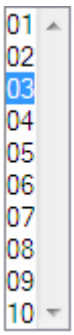


Рис. 3.9. Список для выбора числа

Группа переключателей

```
string radiolist(array $data, string $name, mixed $attribs = null, mixed $optKey
= 'value', string $optText = 'text',
string $selected = null, bool $idtag = false, bool $translate = false)
```

где

$data	- массив объектов;
$name	- имя, общее для всех переключателей;
$attribs	- дополнительные атрибуты тега <input> (могут быть заданы сразу в виде строки);
$optKey	- из какого поля объекта, представляющего собой элемент массива $data, брать значения элементов <input>;
$optText	- из какого поля объекта, представляющего собой элемент массива $data, брать текст элементов <input>;
$selected	- значение выбранного по умолчанию элемента;
$idtag	- префикс id сгенерированных тегов. Для каждого тега <input> id станет равным $idtag<key>, где <key> - значение поля под названием $optKey в соответствующем объекте из массива $data. Если $idtag не задан, то вместо него будет использовано значение $name. В Joomla 1.7 в этой функции имеется ошибка: если объекты массива $data содержат поле с названием id, то к каждому элементу <input> добавляется второй id со значением, взятым из этого поля;
$translate	- пропускать ли текст через функцию JText::_().

Пример:

```
$query = 'SELECT id,title FROM #__content';
$db =& JFactory::getDBO();
$db->setQuery($query);
$content = $db->loadObjectList();
echo JHTML::_('select.radiolist', $content, 'radlist', 'class="inputbox"', 'id',
'title', 2, 'radlist', false);
```

Результат показан на рис. 3.10.

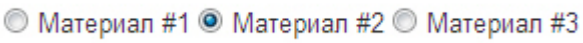


Рис. 3.10. Группа переключателей

Практика

Обработка категорий

В единственном вопросе, который хранится в нашей базе, в поле id_cat стоит значение 1. Оно должно означать id категории. Добавим таблицу и код для работы с категориями.

Создание таблицы для категорий

Выполните следующий SQL-запрос, чтобы создать таблицу для хранения категорий:

```
CREATE TABLE `jos_myquestions_categories`
(
`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
`name` VARCHAR(255) NOT NULL,
`desc` TEXT NOT NULL DEFAULT ''
)
```

Добавьте записи в эту таблицу:

```
INSERT INTO `jos_myquestions_categories` VALUES (NULL,'Без категории',''),
 (NULL,'Риторические вопросы','Вопросы, не требующие ответа')
```

Зайдите в **phpMyAdmin** и убедитесь, что таблица **jos_myquestions_categories** содержит две записи (рис. 3.11).

			id	name	desc
☐	✎	✕	1	Без категории	
☐	✎	✕	2	Риторические вопросы	Вопросы, не требующие ответа

Рис. 3.11. Таблица базы данных, хранящая данные о категориях

Таким образом, сейчас все вопросы, для которых в поле **id_cat** стоит значение 1, относятся к категории "**Без категории**".

Создание класса таблицы

Создайте файл **/administrator/components/com_myquestions/tables/category.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
class TableCategory extends JTable
{
  var $id = null;
  var $name = null;
  var $desc = null;

  function __construct(&$db)
  {
    parent::__construct('#__myquestions_categories', 'id', $db);
  }
}
?>
```

Вывод списка категорий

Добавьте в файл **admin.myquestions.php** функцию showCategories():

```
function showCategories($option)
{
```

```
    $db =& JFactory::getDbo();
    $query = "SELECT * FROM #__myquestions_categories";
    $db->setQuery($query);
    $rows = $db->loadObjectList();
    if ($db->getErrorNum())
    {
        echo $db->stderr();
        return false;
    }
    HTML_questions::showCategories($option, $rows);
}
```

Эта функция аналогична функции showQuestions(), рассмотренной ранее.

В файл **admin.myquestions.html.php** в класс HTML_questions добавьте еще одну функцию:

```
function showCategories($option, &$rows)
{
?>
  <form action="index.php" method="post" name="adminForm">
    <table class="adminlist">
      <thead>
        <tr>
          <th width="20">
            <input type="checkbox" name="toggle" value="" onclick="checkAll(<?
php echo count($rows);?>);"/>
          </th>
          <th class="title" width="30%"><?php echo
JText::_('COM_MYQUESTIONS_CATEGORY_NAME');?></th>
          <th><?php echo JText::_('COM_MYQUESTIONS_CATEGORY_DESC');?></th>
        </tr>
      </thead>
      <?php
        jimport('joomla.filter.output');
        $k = 0;
        for  ($i = 0,  $n = count($rows); $i < $n;  $i ++)
        {
          $row = &$rows[$i];
          $checked = JHTML::_('grid.id', $i, $row->id);
          $link = JFilterOutput::ampReplace('index.php?option=' .$option .
'&task=editcat&cid[]='. $row->id);
      ?>
      <tr class="<?php echo "row$k";?>">
        <td><?=$checked?></td>
        <td><?='<a href="'.$link.'">'.$row->name.'</a>'?></td>
        <td><?=$row->desc?></td>
      </tr>
      <?php
        $k = 1 - $k;
      }
      ?>
    </table>
    <input type="hidden" name="option" value="<?php echo $option;?>"/>
    <input type="hidden" name="task" value=""/>
    <input type= "hidden" name="boxchecked" value="0"/>
  </form>
<?php
}
```

Данная функция также аналогична одноименной функции для отображения списка вопросов.

Добавьте в переключатель switch в файле **admin.myquestions.php** обработку новой задачи:

```
case 'showcat':
   showCategories($option);
   break;
```

Добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** код:

```
COM_MYQUESTIONS_CATEGORY_NAME="Название категории"
COM_MYQUESTIONS_CATEGORY_DESC="Описание категории"
```

Перейдя по ссылке http://localhost/joomla/administrator/index.php?option=com_myquestions&task=showcat, вы уже можете увидеть список категорий.

Однако пока над списком будет отображаться старая панель инструментов, которую мы создали для списка вопросов. Поэтому необходимо также создать новые панели инструментов для работы с категориями. Откройте файл **toolbar.myquestions.php** и добавьте в переключатель switch следующий код:

```
case 'showcat':
  TOOLBAR_myquestions_categories::_DEFAULT();
  break;
```

Как видите, мы добавили обработку задачи showcat - отображение списка категорий. Соответственно, вызывается функция _DEFAULT() класса TOOLBAR_myquestions_categories. Напишем код этого класса. Добавьте в файл **toolbar.myquestions.html.php** код:

```
class TOOLBAR_myquestions_categories
{
  function _DEFAULT()
  {
    JToolBarHelper::title(JText::_('COM_MYQUESTIONS_TOOLBAR_TITLE_CATEGORIES'),
     'generic.png');
    JToolBarHelper::addNew('addcat');
    JToolBarHelper::editList('editcat');     JToolBarHelper::deleteList
    (JText::_('COM_MYQUESTIONS_TOOLBAR_REMOVE_CATEGORIES_CONFIRMATION'),
     'removecat');  }
}
```

Таким образом, панель инструментов для списка категорий будет содержать три кнопки: "**Создать**", "**Изменить**" и "**Удалить**".

Добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** код:

```
COM_MYQUESTIONS_TOOLBAR_TITLE_CATEGORIES="Управление категориями
вопросов"
COM_MYQUESTIONS_TOOLBAR_REMOVE_CATEGORIES_CONFIRMATION="Вы
действительно хотите удалить эти категории?"
```

Теперь по ссылке http://localhost/joomla/administrator/index.php?option=com_myquestions&task=showcat видим список категорий (рис. 3.12).

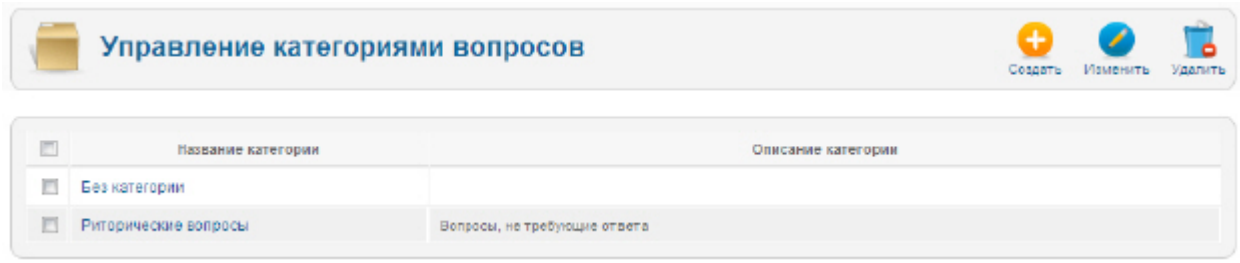


Рис. 3.12. Список категорий в бэкенде

Создание, редактирование и удаление категорий

При нажатии на кнопки "**Создать**" или "**Изменить**", расположенные над списком категорий, должна отображаться другая панель инструментов. Для этого добавьте в класс TOOLBAR_myquestions_categories функцию _NEW():

```
class TOOLBAR_myquestions_categories
{
  function _NEW()
  {
    JToolBarHelper::title(JText::_('COM_MYQUESTIONS_TOOLBAR_TITLE_CATEGORIES'),
     'generic.png');
    JToolBarHelper::save('savecat');
    JToolBarHelper::apply('applycat');
    JToolBarHelper::cancel('showcat');
  }
  function _DEFAULT()
  {
    JToolBarHelper::title(JText::_('COM_MYQUESTIONS_TOOLBAR_TITLE_CATEGORIES'),
     'generic.png');
    JToolBarHelper::addNew('addcat');
    JToolBarHelper::editList('editcat');

JToolBarHelper::deleteList(JText::_('COM_MYQUESTIONS_TOOLBAR_REMOVE_CONFIRMATION
'),
      'removecat');
  }
}
```

Запишем в файле **toolbar.myquestions.php**, что при обработке задач addcat и editcat должна отображаться панель инструментов _NEW. Измените код этого файла так:

```
<?php
defined('_JEXEC') or die('Restricted access');
require_once(JApplicationHelper::getPath('toolbar_html'));
switch($task)
{
  case 'reply':
    TOOLBAR_myquestions::_REPLY();
    break;
  default:
    TOOLBAR_myquestions::_DEFAULT();
    break;
  case 'showcat':
    TOOLBAR_myquestions_categories::_DEFAULT();
```

```
    break;
  case 'addcat':
  case 'editcat':
    TOOLBAR_myquestions_categories::_NEW();
    break;
}
?>
```

Теперь добавим обработку всех перечисленных в файле **toolbar.myquestions.html.php** задач. Откройте файл **admin.myquestions.php** и добавьте в переключатель switch код:

```
case 'addcat':
case 'editcat':
    editCategory($option);
    break;
case 'savecat':
case 'applycat':
    saveCategory($option, $task);
    break;
case 'removecat':
    removeCategories($option);
    break;
```

Добавьте перечисленные функции в файл **admin.myquestions.php**:

```
function editCategory($option)
{
    $row =& JTable::getInstance('Category','Table');
    $cid = JRequest::getVar('cid', array(0), '', 'array');
    $id = $cid[0];
    $row->load($id);
    HTML_questions::editCategory($row, $option);
}
function saveCategory($option, $task)
{
    $row =& JTable::getInstance('category', 'Table');
    if (!$row->bind(JRequest::get('post')))
    {
        echo "<script> alert('".$row->getError()."');
          window.history.go(-1); </script>\n";
        exit();
    }
    $row->desc = JRequest::getVar('desc', '', 'post', 'string',
JREQUEST_ALLOWRAW);

    if (!$row->store())
    {
        echo "<script> alert('".$row->getError()."');
         window.history.go(-1); </script>\n";
        exit();
    }

    global $app;
    if ($task == 'savecat')
        $app->redirect('index.php?option='.$option.'&task=showcat',
         JText::_('COM_MYQUESTIONS_CATEGORY_SAVED'));
    else
        if ($task == 'applycat')
```

```
            $app-
            >redirect('index.php?option='.$option.'&task=editcat&cid[]='.$row-
            >id, JText::_('COM_MYQUESTIONS_CATEGORY_SAVED'));
    }
    function removeCategories($option)
    {
        global $app;
        $cid = JRequest::getVar('cid', array(), '', 'array');
        $db =& JFactory::getDbo();
        if(count($cid))
        {
            $cids = implode(',', $cid);
            $query = "DELETE FROM #__myquestions_categories WHERE id IN
($cids)";
            $db->setQuery($query);
            if (!$db->query())
            {
                echo "<script> alert('".$db->getErrorMsg()."');
                 window.history.go(-1); </script>\n";
            }
        }
        $app->redirect('index.php?option=' . $option . '&task=showcat',
         JText::_('COM_MYQUESTIONS_CATEGORY_DELETED'));
    }
```

Листинг .

Добавьте в файл **admin.myquestions.html.php** в класс HTML_questions метод editCategory() для отображения формы редактирования категории:

```
function editCategory ($row, $option)
{
  $editor =& JFactory::getEditor();
  ?>
  <form action = "index.php" method="post" name="adminForm"
    id="adminForm">
    <fieldset class="adminform">
      <table class="admintable" width=100%>
        <tr>
          <td width="100" class="key">
              <?php echo JText::_('COM_MYQUESTIONS_CATEGORY_NAME');?>:
          </td>
          <td>
            <input class="text_area" type="text" name="name" id="name"
             size="50" maxlength="255" value="<?php echo $row->name;?>"/>
          </td>
        </tr>
        <tr>
          <td width="100" class="key">
            <?php echo JText::_('COM_MYQUESTIONS_CATEGORY_DESC');?>:
          </td>
          <td>
            <?php
              echo $editor->display('desc',  $row->desc,'100%', '250', '40',
'10');?>
          </td>
        </tr>
      </table>
```

```
    </fieldset>
    <input type="hidden" name="id" value="<?php echo $row->id;?>"/>
    <input type="hidden" name="option" value="<?php echo $option;?>"/>
    <input type="hidden" name="task" value=""/>
  </form>
<?php
}
```

Наконец, добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** код:

```
COM_MYQUESTIONS_CATEGORY_SAVED="Категория сохранена"
COM_MYQUESTIONS_CATEGORY_DELETED="Категории успешно удалены"
```

Убедитесь, что все кнопки обеих панелей инструментов работают корректно.

Присвоение вопросу какой-либо категории

Добавим в форму ответа на вопрос выпадающий список для выбора категории.

Откройте файл **admin.myquestions.php** и измените код функции replyToQuestion() следующим образом:

```
function replyToQuestion($option)
{
    $row =& JTable::getInstance('Question','Table');
    $cid = JRequest::getVar('cid', array(0), '', 'array');
    $id = $cid[0];
    $row->load($id);
    $db = &JFactory::getDBO();
    $query = 'SELECT name AS text, id AS value FROM #__myquestions_categories';
    $db->setQuery($query);
    $categories = $db->loadObjectList();
    $list_cat = JHTML::_('select.genericlist',  $categories,  'id_cat',
    ' class="inputbox" ', 'value', 'text', $row->id_cat);
    HTML_questions::replyToQuestion($row, $option, $list_cat);
}
```

В файле **admin.myquestions.html.php** измените прототип функции HTML_questions::replyToQuestion() так:

```
function replyToQuestion ($row, $option, $list_cat)
```

Наконец, в коде этой функции измените фрагмент, в котором раньше выводилось текстовое поле с id категории, так:

```
<tr>
  <td width="100" class="key">
    <?php echo JText::_('COM_MYQUESTIONS_CATEGORY');?>:
    </td>
  <td>
    <?=$list_cat?>
  </td>
</tr>
```

Перейдите к странице ответа на вопрос и убедитесь, что отображается выпадающий список для выбора категории (рис. 3.13).

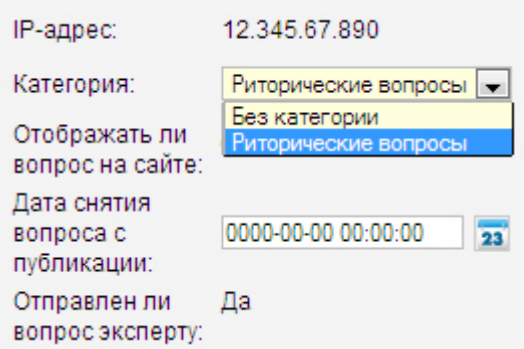


Рис. 3.13. Выбор категории вопроса в бэкенде

Ключевые термины

JHTML	- класс для вывода элементов XHTML.
JHTMLBehavior	- поддерживающий класс, который позволяет вывести календарь, дерево элементов, файловый загрузчик и некоторые другие элементы управления.
JHTMLEmail	- поддерживающий класс, содержащий метод для скрытия адреса электронной почты в целях его защиты от спам-ботов.
JHTMLForm	- поддерживающий класс, содержащий метод, который возвращает код скрытого поля формы для уменьшения риска CSRF-атак.
JHTMLGrid	- поддерживающий класс, позволяющий вывести в таблице в панели управления такие элементы, как чекбокс, пиктограмма для переключения состояния "опубликовано"/"не опубликовано", отобразить заголовок столбца как ссылки для сортировки по этому столбцу и др.
JHTMLImage	- поддерживающий класс, содержащий методы для поиска изображения в фронтенде и бэкенде.
JHTMLList	- поддерживающий класс для создания списков некоторых конкретных значений.
JHTMLSelect	- поддерживающий класс для генерации кода списков.
Основной метод класса JHTML	- метод JHTML::_(), который вызывает метод, определяющийся его первым параметром, и передает ему свои остальные параметры.
Поддерживающие классы	- классы для вывода элементов XHTML и поведений Javascript.

Краткие итоги

Joomla содержит методы для генерации и отображения элементов XHTML и поведений JavaScript. Эти методы вызываются с помощью метода JHTML::_(). По первому параметру данный метод определяет, какой метод необходимо вызвать, а остальные параметры передаются в этот метод. Таким образом могут быть вызваны как методы самого класса JHTML, так и методы поддерживающих классов.

Методы класса JHTML позволяют вывести следующие элементы: календарь, форматированную дату, элементы <iframe>, <img>, <a>, <script>, <link> и всплывающую подсказку.

Поддерживающие классы используются следующим образом:

- JHTMLBehavior позволяет вывести календарь, дерево элементов, файловый загрузчик и некоторые другие элементы управления.
- JHTMLEmail содержит метод для скрытия адреса электронной почты в целях его защиты от спам-ботов.
- JHTMLForm содержит метод, который возвращает код скрытого поля формы для уменьшения риска CSRF-атак.
- JHTMLGrid позволяет вывести в таблице в панели управления такие элементы, как чекбокс, пиктограмма для переключения состояния "опубликовано"/"не опубликовано", отобразить заголовок столбца как ссылки для сортировки по этому столбцу и др.
- JHTMLImage содержит два метода для поиска изображения в фронтенде и бэкенде соответственно.
- JHTMLList используется для создания списков некоторых конкретных значений.
- JHTMLSelect используется для генерации кода списков.

Вопросы

1. Каким образом работает метод JHTML::_()?
2. Какие элементы могут быть отображены с помощью класса JHTML?
3. Для чего используется класс JHTMLBehavior?
4. Для чего используется класс JHTMLEmail?
5. Для чего используется класс JHTMLForm?
6. Код каких элементов генерируют методы класса JHTMLGrid?
7. В чем преимущество использования класса JHTMLImage?
8. Для чего используются классы JHTMLList и JHTMLSelect?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. список вариантов в дополнительных материалах).

4. Лекция: Иерархия пунктов меню. Отправка писем. Классы ядра JEditor, JURI, JError, Jdate

Рассмотрен принцип хранения иерархии пунктов меню, использующийся в Joomla. Рассмотрены классы для отправки электронной почты, отображения кода визуального редактора, работы с URI, ошибками и датами.

Цель лекции: Ознакомиться со способом программного добавления пунктов меню в Joomla, а также с некоторыми методами классов JMail, JEditor, JURI, JError, JDate.

Организация иерархии пунктов меню в Joomla

Для хранения иерархии пунктов меню в Joomla используются вложенные множества (http://en.wikipedia.org/wiki/Nested_set_model). Например, на рис. 4.1 представлена эта иерархия для случая, если бы в ней было всего восемь пунктов: корневой пункт меню **Menu_Item_Root**, **com_banners** с подпунктами **com_banners**, **com_banners_categories**, **com_banners_clients** и **com_messages** с подпунктами **com_messages_add**, **com_messages_read**:

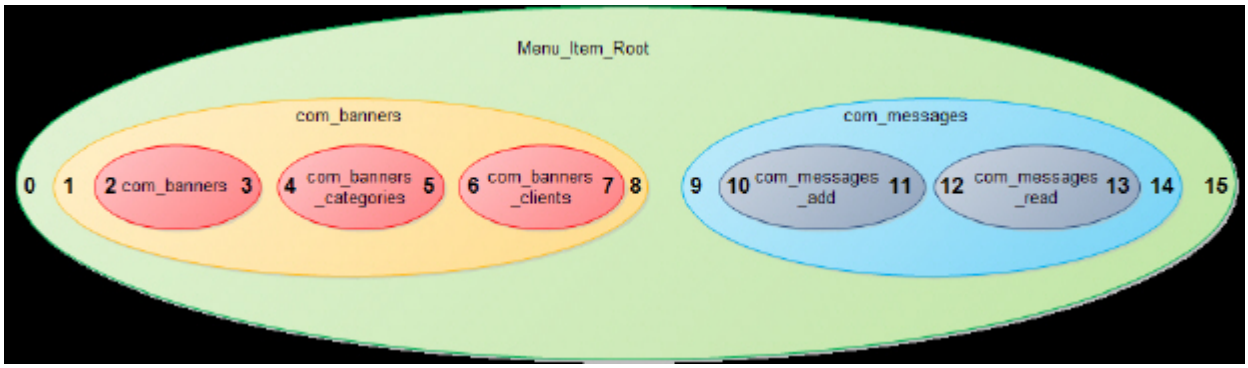


Рис. 4.1. Пример использования вложенных множеств для организации иерархии пунктов меню

Каждому элементу иерархии соответствуют два числа - левый и правый ключи.

Для каждого пункта в таблице **#__menu** хранится **id** родителя **parent_id**, уровень **level** и левый и правый ключи **lft** и **rgt**. На рис. 4.2 оранжевым цветом показаны **id**, синим - уровни, красным - левые и правые ключи.

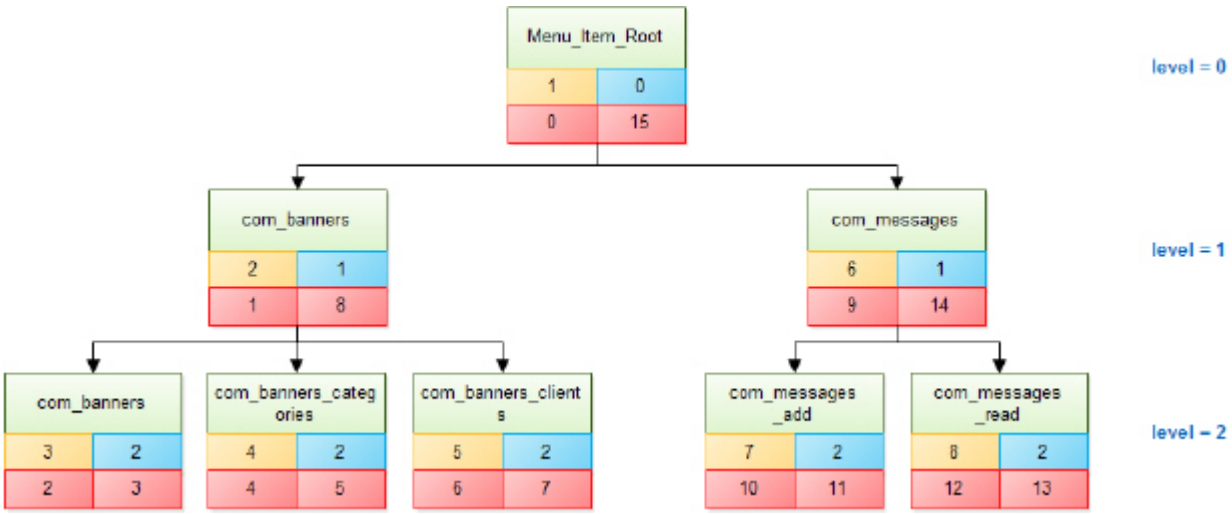


Рис. 4.2. Данные, хранящиеся для каждого пункта меню как элемента иерархии

Отправка писем (класс JMail)

Класс JMail предназначен для создания и отправки электронных писем. JMail поддерживает три механизма отправки почты: функцию PHP mail(), Sendmail и SMTP.

Доступ к глобальному объекту JMail можно получить через метод getMailer() класса JFactory:

```
$mailer =& JFactory::getMailer();
```

Отправитель письма

Для задания отправителя письма используется метод

```
JMail setSender(mixed $from)
```

где $from - либо строка, содержащая e-mail отправителя, либо массив, который содержит e-mail и имя отправителя.

Примеры:

```
// Первый вариант
$mailer =& JFactory::getMailer();
$mailer->setSender('vasya@mysite.ru');

// Второй вариант
$sender = array('vasya@mysite.ru', 'Вася')
$mailer->setSender($sender);
```

Правильнее будет не задавать эти значения вручную, а получать их из настроек отправки почты, заданных в панели управления (**"Сайт" - "Общие настройки" - "Сервер"**):

```
$config =& JFactory::getConfig();
$sender = array(
    $config->getValue('config.mailfrom'),
    $config->getValue('config.fromname'));
$mailer->setSender($sender);
```

Адреса для ответа

Адреса для ответа задаются методом

```
JMail addReplyTo(array $replyto)
```

где $replyto - пара e-mail-имя или массив таких пар.

Например:

```
// Первый вариант
$reply = array('vasya@mysite.ru', 'Вася');
$mailer->addReplyTo($reply);


// Второй вариант
$reply0 = array('vasya@mysite.ru', 'Вася');
$reply1 = array('vanya@mysite.ru', 'Ваня');
$replies = array($reply0, $reply1);
$mailer->addReplyTo($replies);
```

Получатель

Для задания получателя, получателя копии (Carbon Copy) и получателя скрытой копии (Blind Carbon Copy), используются методы

```
JMail addRecipient(mixed $recipient, mixed name = '')
JMail addCC(mixed $cc, mixed name = '')
JMail addBCC(mixed $bcc, mixed name = '')
```

где

$recipient, $cc, $bcc	- e-mail или массив, состоящий из e-mail;
$name	- имя получателя или массив, состоящий из имен.

Для примера зададим в качестве адреса получателя адрес текущего пользователя:

```
$user =& JFactory::getUser();
$mailer->addRecipient($user->email);
```

Задание нескольких получателей:

```
$recipient = array('vasya@mysite.ru', 'vanya@mysite.ru');
$mailer->addRecipient($recipient);
```

Тема, текст, вложения

Тема, текст и вложения письма задаются методами

```
JMail setSubject(string $subject)
JMail setBody(string $content)
JMail addAttachment(mixed $attachment, mixed $name = '', mixed $encoding =
'base64', mixed $type = 'application/octet-stream')
```

где

$subject	- строка, содержащая тему письма;
$content	- строка, содержащая текст письма;
$attachment	- строка, содержащая путь к файлу, или массив таких строк;
$name	- строка, содержащая имя файла, или массив таких строк;
$encoding	- кодировка сообщения: 8bit, 7bit, binary, base64, quoted-printable;
$type	- MIME-тип файла.

Например:

```
$mailer->setSubject('Тема письма');
$mailer->setBody("Текст письма");
$mailer->addAttachment(JPATH_COMPONENT.DS.'files'.DS.'document.pdf');
```

По умолчанию содержимое письма имеет формат **plain text**. Если вам нужно отправить письмо в формате HTML, используйте метод IsHTML(), задающий MIME-тип содержимого письма как text/html:

```
$mailer->IsHTML(true);
$mailer->setBody("Текст письма, содержащий <b>HTML-теги</b>");
```

В этом случае желательно указать кодировку письма - base64, чтобы в нем не появились нежелательные символы:

```
$mailer->Encoding = 'base64';
```

Отправка

Наконец, для отправки письма используется метод Send(), возвращающий true при успешной отправке или объект JError в случае ошибки:

```
if ($mailer->Send() !== true)
    die('Ошибка отправки письма');
```

Проверка корректности e-mail

Помимо класса JMail, существует статический класс JMailHelper. Большинство его методов предназначены для очистки данных перед добавлением к электронному письму и используются классом JMail. Интерес для разработчика представляет метод isEmailAddress(), который проверяет, является ли заданная строка корректным адресом электронной почты:

```
if (!JMailHelper::isEmailAddress($str))
    die('Недопустимый формат e-mail');
```

Перед использованием класса JMailHelper необходимо подключить библиотеку JMail:

```
jimport('joomla.utilities.mail');
```

WYSIWYG-редактор (класс JEditor)

Класс JEditor используется для работы с WYSIWYG-редактором.

Получить ссылку на глобальный объект JEditor можно так:

```
$editor =& JFactory::getEditor();
```

Отображение редактора

Для отображения выбранного администратором сайта редактора используется метод display(). Если не выбран ни один редактор, то будет отображено поле <textarea>.

```
string display(string $name, string $html, string $width, string $height, int
$col, int $row, bool $buttons = true,
string $id = null, string $asset = null, object $author = null, array $params =
array())
```

где

$name	- имя элемента управления формы;
$html	- содержимое поля;
$width	- ширина текстового поля в процентах или пикселях;
$height	- высота текстового поля;
$col	- количество столбцов в поле <textarea> (применяется, если администратор выбрал не использовать HTML-редактор);
$row	- количество строк в <textarea>;
$buttons	- отображать ли кнопки редактора ("**Материал**", "**Изображение**", "**Разрыв страницы**", "**Подробнее…**");
$id	- **id** элемента управления формы;
$asset и $author	- используются для подтверждения прав доступа пользователя к некоторым функциям плагина редактора. Более подробная информация об этих параметрах в справочной документации отсутствует;
$params	- ассоциативный массив параметров редактора.

Пример:

```
echo $editor->display('question',  $row->question,'100%', '250', '40', '10');
```

Если по умолчанию задан редактор TinyMCE, то этот код выведет на экран следующее (рис. 4.3).

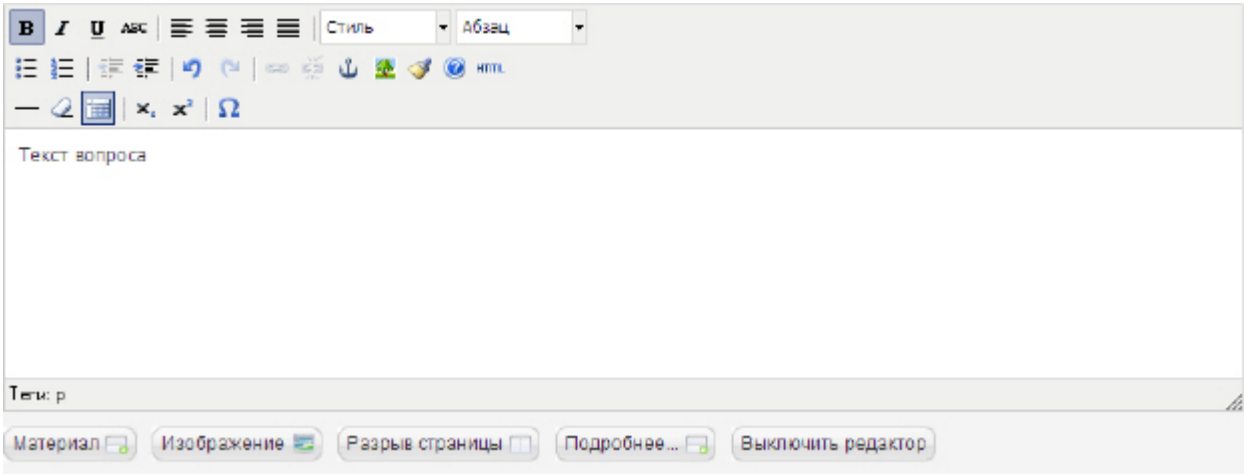


Рис. 4.3. Отображение WYSIWYG-редактора

URI (класс JURI)

Чтобы создать новый объект класса JURI или получить ссылку на глобальный объект, используется метод

```
JURI &getInstance(string $uri='SERVER')
```

где $uri - URI страницы. При $uri='SERVER' метод вернет ссылку на глобальный объект, который представляет URI текущей страницы. В противном случае будет создан новый объект.

Например, так можно получить URI текущей страницы:

```
$u = &JURI::getInstance();
echo $u->toString();
```

Методы класса JURI

Методы класса JURI хорошо документированы (http://docs.joomla.org/JURI), поэтому здесь мы только перечислим некоторые из них.

Получение базового URI сайта:

```
string base(bool $pathonly)
```

где $pathonly - возвращать только путь или также вернуть хост и порт.

Построение запроса (т.е. компонента URI):

```
string buildQuery(array $params)
```

где $params - ассоциативный массив пар "**имя-значение**".

Получение текущего URI без запроса и фрагмента:

```
string current()
```

Получение отдельных компонентов URI:

фрагмента	- string getFragment();
хоста	- string getHost();
пароля	- string getPass();
пути	- string getPath();
порта	- string getPort();
запроса	- string getQuery(bool $toArray), где $toArray задает, возвращать ли элементы запроса в виде ассоциативного массива или в виде строки;
схемы	- string getScheme();
имени пользователя	- string getUser().

Получение заданной переменной из запроса:

```
string getVar(string $name, string $default)
```

где

$name	- имя переменной;
$default	- значение по умолчанию.

Заполнение полей класса значениями, полученными из заданного URI ($uri):

```
boolean parse(string $uri)
```

Получение корневого URI сайта:

```
string root(bool $pathonly, string $path)
```

где $path - задает новый путь.

Ошибки (класс JError)

Методы raiseError(), raiseNotice() и raiseWarning() класса JError вызываются при возникновении каких-либо ошибок. Все они возвращают объект JException, содержащий сведения об ошибке, например, в каких файлах она возникла. raiseError() добавляет заданное сообщение в очередь сообщений и останавливает выполнение расширения, а остальные два метода только добавляют сообщение в очередь, но работа расширения продолжается.

```
JException raiseError(string $code, string $msg, mixed $info = null)
JException raiseWarning(string $code, string $msg, mixed $info = null)
JException raiseNotice(string $code, string $msg, mixed $info = null)
```

где

$code	- внутренний код ошибки, задается по усмотрению программиста;
$msg	- сообщение об ошибке для пользователя;
$info	- дополнительная информация об ошибке для программиста.

Пример:

```
if ($this->published == 0)
  JError::raiseError (404, "Нет такой страницы");
```

Перечисленные методы создавались в то время, когда Joomla работала с PHP4, где отсутствовал механизм обработки исключений. Если на хостинге стоит PHP5, предпочтительнее использовать класс Exception.

Даты (класс JDate)

JDate - класс для работы с датами. Для создания объекта JDate используется метод getDate() класса JFactory:

```
JDate JFactory::getDate(mixed $time = 'now', mixed $tzOffset = null)
```

где

$time	- дата и время в формате, годном для функции PHP strtotime();
$tzOffset	- временная зона. Может задаваться числом от -12 до 14 (смещение относительно всемирного координированного времени - UTC) или строкой из числа временных зон, поддерживаемых PHP (их список можно найти на странице http://www.php.net/manual/ru/timezones.php).

В отличие от многих других методов JFactory, getDate() не возвращает глобальный объект, а создает новый. Поэтому перед вызовом этого метода не нужно ставить амперсанд:

```
$now = JFactory::getDate(); // текущие дата и время
$some_date = JFactory::getDate('2012-01-01 00:00:00','Europe/Moscow');
```

Получение даты в каком-либо формате

```
string format(string $format, bool $local = false, bool $translate =
true)
```

где

$format	- формат;
$local	- возвращать дату в локальной временной зоне или в GMT;
$translate	- переводить ли текст.

Чтобы не задавать формат даты вручную, можно использовать один из определенных в Joomla форматов: DATE_FORMAT_LC, DATE_FORMAT_LC1 и т.д. Их список находится в языковом файле **language/<ln-LN>/<ln-LN>.ini**, где **<ln-LN>** - код языка. Как и для всякого языкового ключа, значение такого формата можно получить с помощью функции JText::_(). Например, выведем дату в формате DATE_FORMAT_LC3:

```
echo $some_date->format(JText::_('DATE_FORMAT_LC3'));
```

Получение даты в формате, пригодном для вставки в запрос SQL

```
string toMySQL(bool $local=false)
```

Например, для получения материалов, созданных ранее даты $some_date, можно выполнить запрос:

```
$query = "SELECT * FROM #__content WHERE created < '{$some_date->toMySQL()}'";
```

Практика

Отправка уведомлений по электронной почте

Сейчас в форме для ответа на вопрос две кнопки на панели инструментов (рис. 4.4) - нерабочие.

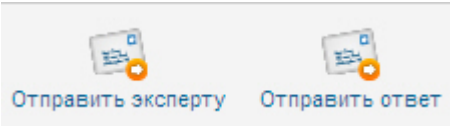


Рис. 4.4. Кнопки для отправки уведомлений по электронной почте

Напишем код для их обработки.

Откройте файл **admin.myquestions.php** и добавьте в переключатель switch обработку двух задач:

case 'sendToExpert':

```
case 'sendAnswer':
    send($option,$task);
    break;
```

Добавьте также в этот же файл функцию send():

```
function send($option,$task)
{
    $row_new = save();
    $q = $row_new->question;
    $a = $row_new->answer;

    $mailer =& JFactory::getMailer();
    $mailer->setSender('test@mysite.ru');
    if ($task == 'sendToExpert')
    {
        $mailer->addRecipient('expert@mysite.ru');
        $mailer->setSubject(JText::_('COM_MYQUESTIONS_NEW_QUESTION'));
        $mailer->setBody(JText::sprintf('COM_MYQUESTIONS_EMAIL_EXPERT_BODY',
$q));
    }
    else
    {
        $mailer->addRecipient($row_new->email);
        $mailer->setSubject(JText::_('COM_MYQUESTIONS_NEW_ANSWER'));
        $mailer->setBody(JText::sprintf('COM_MYQUESTIONS_EMAIL_USER_BODY',$q,
$a));
    }
    $mailer->IsHTML(true);

    if ($mailer->Send() !== true)
        $message = 'COM_MYQUESTIONS_EMAIL_ERROR';
    else
    {
        $message = 'COM_MYQUESTIONS_EMAIL_SUCCESS';

        $db =& JFactory::getDbo();
        if ($task == 'sendToExpert')
            $query = "UPDATE #__myquestions SET senttoexpert=1 WHERE
id={$row_new->id}";
        else
            $query = "UPDATE #__myquestions SET senttoauthor=1 WHERE
id={$row_new->id}";

        $db->setQuery($query);
        $db->query();
        if ($db->getErrorNum())
        {
            echo $db->stderr();
            return false;
        }
    }
    global $app;
    $app->redirect('index.php?option='.$option.'&task=reply&cid[]='.$row_new-
>id, JText::_($message));
}
```

Листинг .

В данной функции прежде всего происходит сохранение текущей записи и обновленная запись сохраняется в переменной row_new. Затем в зависимости от того, отправляется это уведомление эксперту или автору вопроса, задаются отправитель, получатель, тема и текст

письма. Если уведомление отправляется пользователю, задавшему вопрос, то адрес электронной почты получателя берется из записи row_new.

При задании тела письма используется метод JText::sprintf(), пропускающий заданную ему строку через PHP-функцию sprintf(). В выражении
JText::sprintf('COM_MYQUESTIONS_EMAIL_USER_BODY',$q,$a)
'COM_MYQUESTIONS_EMAIL_USER_BODY' - это строка формата, которую мы сейчас зададим в языковом файле, а $q и $a - это аргументы (соответственно текст вопроса и текст ответа).

Тип содержимого письма задается как text/html, так как оно содержит html-теги.

Если письмо успешно отправлено, то в соответствующей вопросу записи в базе данных значение senttoexpert или senttoauthor устанавливается равным 1.

В конце функции происходит перенаправление на текущую страницу с выводом сообщения либо об успешной отправке письма, либо об ошибке.

Добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** следующий код:

```
COM_MYQUESTIONS_EMAIL_SUCCESS="e-mail отправлен"
COM_MYQUESTIONS_EMAIL_ERROR="Не удалось отправить e-mail"
COM_MYQUESTIONS_NEW_QUESTION="Новый вопрос"
COM_MYQUESTIONS_EMAIL_EXPERT_BODY="<p>Добрый день!</p><p>На сайте появился новый
вопрос:
</p><p><i>%s</i></p>"
COM_MYQUESTIONS_NEW_ANSWER="Ответ на ваш вопрос"
COM_MYQUESTIONS_EMAIL_USER_BODY="<p>Добрый день!</p><p>На сайте появился ответ
на ваш вопрос:
</p><p><i>%s</i></p><p>Ответ был таким:</p><p><i>%s</i></p>"
```

Как видите, строки COM_MYQUESTIONS_EMAIL_EXPERT_BODY и COM_MYQUESTIONS_EMAIL_USER_BODY содержат параметры. Так как их тип - строковый (тексты вопроса, ответа), то используется описатель типа %s.

Теперь перейдите в бэкенде на страницу ответа на вопрос и нажмите кнопку "**Отправить эксперту**". Вы должны увидеть сообщение о том, что письмо отправлено (рис. 4.5).

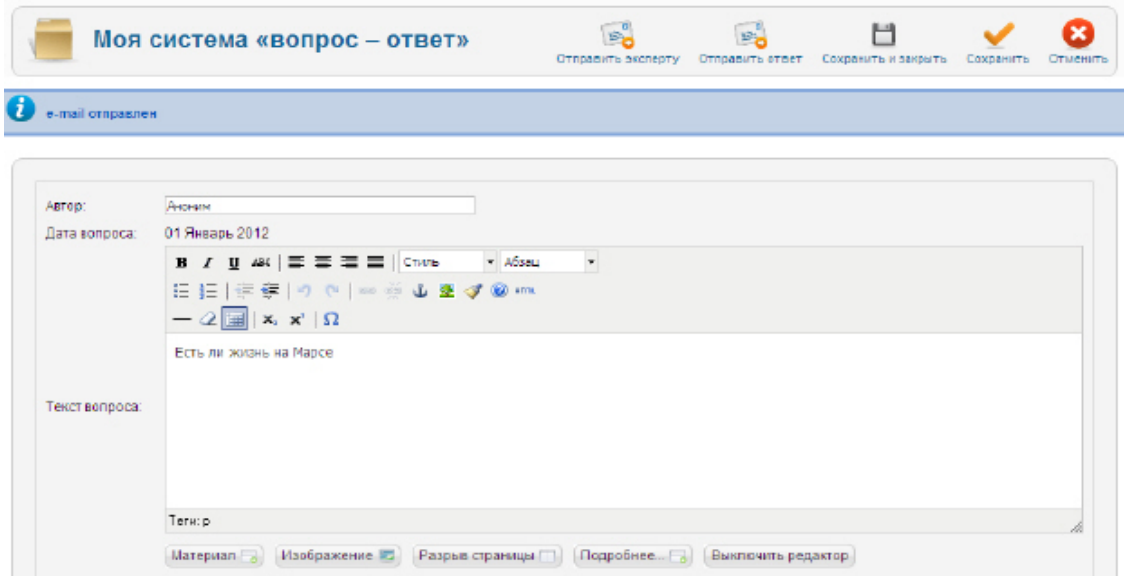


Рис. 4.5. Сообщение об отправке уведомления по электронной почте

Если вы используете локальный сервер, то ваше письмо, скорее всего, на самом деле не отправилось. В Денвере срабатывает почтовая заглушка sendmail: все письма просто помещаются в папку **<путь к Денверу>/tmp/!sendmail** в виде файлов с расширением .eml. Если в вашем случае так и есть, то откройте эту папку и найдите в ней только что "отправленное" письмо (можно ориентироваться по названию файла - это дата и время "отправки"). Откройте файл в почтовой программе и убедитесь, что его содержимое соответствует заданным нами значениям (рис. 4.6).

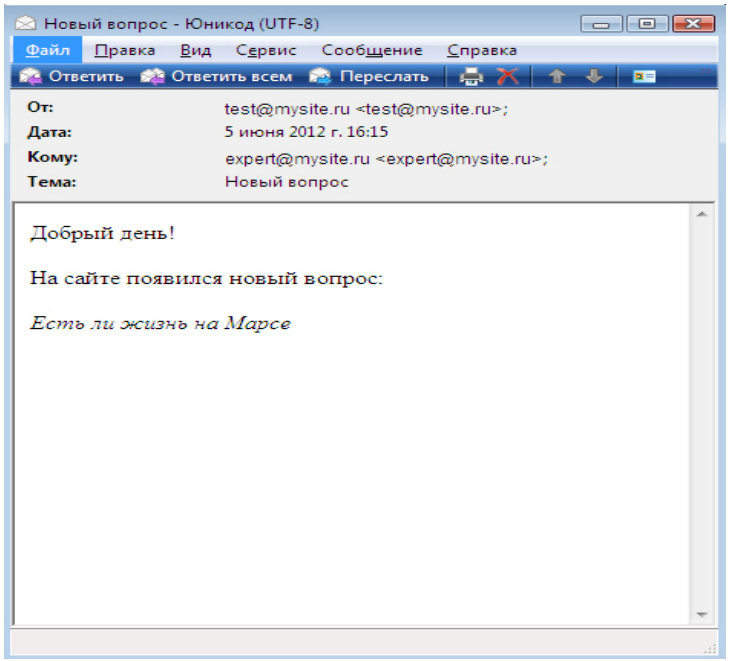


Рис. 4.6. Результат отправки уведомления на локальном сервере - файл .eml, открытый в почтовой программе

Перейдите к списку вопросов и убедитесь, что значение в столбце "**Отправлен ли вопрос эксперту**" поменялось с "**Нет**" на "**Да**" (рис. 4.7).

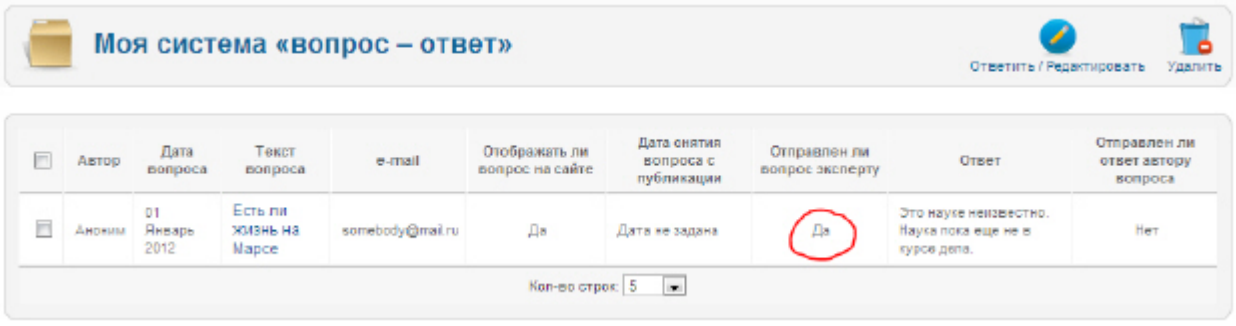


Рис. 4.7. Новое значение в столбце "Отправлен ли вопрос эксперту"

Аналогично проверьте, как работает кнопка "**Отправить ответ**".

Добавление пункта меню

Добавим пункт меню для управления категориями.

Создавая первый пункт меню для нашего компонента, мы не указали значения левого и правого ключей. Определим их сейчас. Уровень данного пункта равен 1 (его предком является корень дерева с уровнем 0, а потомки могут иметь уровень 2 и более). Родительский узел - это самая первая строка в таблице #__menu, посмотрите значение его правого ключа в поле rgt.

Пусть $right_key - правый ключ родительского узла, $level - уровень родительского узла. Тогда для задания правильных значений левого и правого ключей выполним следующие запросы (не забудьте заменить $right_key и $level на значения из вашей таблицы):

1. Обновление ключей узлов, стоящих за родительским узлом:

   ```
   UPDATE jos_menu SET lft=lft+2, rgt=rgt+2 WHERE lft>$right_key
   ```

2. Обновление родительской ветки:

   ```
   UPDATE jos_menu SET rgt=rgt+2 WHERE rgt>=$right_key AND lft<$right_key
   ```

3. Обновление нового узла:

   ```
   UPDATE jos_menu SET lft=$right_key, rgt=$right_key+1, level=$level+1 WHERE
   title='com_myquestions_menu'
   ```

Пусть теперь $right_key и $level - соответственно правый ключ и уровень этого только что обновленного нами узла, а $parent_id - его id. Создадим два подпункта этого пункта меню (не забудьте заменить 10006 на id вашего компонента из таблицы #__extensions).

Подпункт для управления списком вопросов:

```
UPDATE jos_menu SET lft=lft+2, rgt=rgt+2 WHERE lft>$right_key;
UPDATE jos_menu SET rgt=rgt+2 WHERE rgt>=$right_key AND lft<$right_key;
INSERT INTO `jos_menu` (`menutype`, `title`, `alias`, `path`, `link`, `type`,
`parent_id`, `level`,
`component_id`, `access`, `img`,  `params`, `lft`, `rgt`, `client_id`) VALUES
('menu', 'com_myquestions_menu_questions',
'Questions', 'My Questions/Questions', 'index.php?option=com_myquestions',
'component',  $parent_id, $level+1, 10006, 1,
'class:component', '', $right_key, $right_key+1, 1);
```

Обратите внимание, что значения alias и path соответствуют иерархии узлов.

Посмотрите в таблице #__menu новое значение $right_key главного пункта меню для нашего компонента - оно должно было увеличиться на 2. Теперь создадим подпункт для управления списком категорий:

```
UPDATE jos_menu SET lft=lft+2, rgt=rgt+2 WHERE lft>$right_key;
UPDATE jos_menu SET rgt=rgt+2 WHERE rgt>=$right_key AND lft<$right_key;
INSERT INTO `jos_menu` (`menutype`, `title`, `alias`, `path`, `link`, `type`,
`parent_id`, `level`,
`component_id`, `access`, `img`,  `params`, `lft`, `rgt`, `client_id`)
VALUES ('menu', 'com_myquestions_menu_categories',
'Categories', 'My Questions/Categories', 'index.php?
```

```
option=com_myquestions&task=showcat', 'component', $parent_id,
$level+1, 10006, 1, 'class:component', '', $right_key, $right_key+1, 1);
```

Наконец, добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.sys.ini** код:

```
COM_MYQUESTIONS_MENU_QUESTIONS="Управление вопросами"
COM_MYQUESTIONS_MENU_CATEGORIES="Управление категориями"
```

Обновите любую страницу в бэкенде и убедитесь, что появились два новых подпункта меню "**Моя система "вопрос - ответ"**" (рис. 4.8).

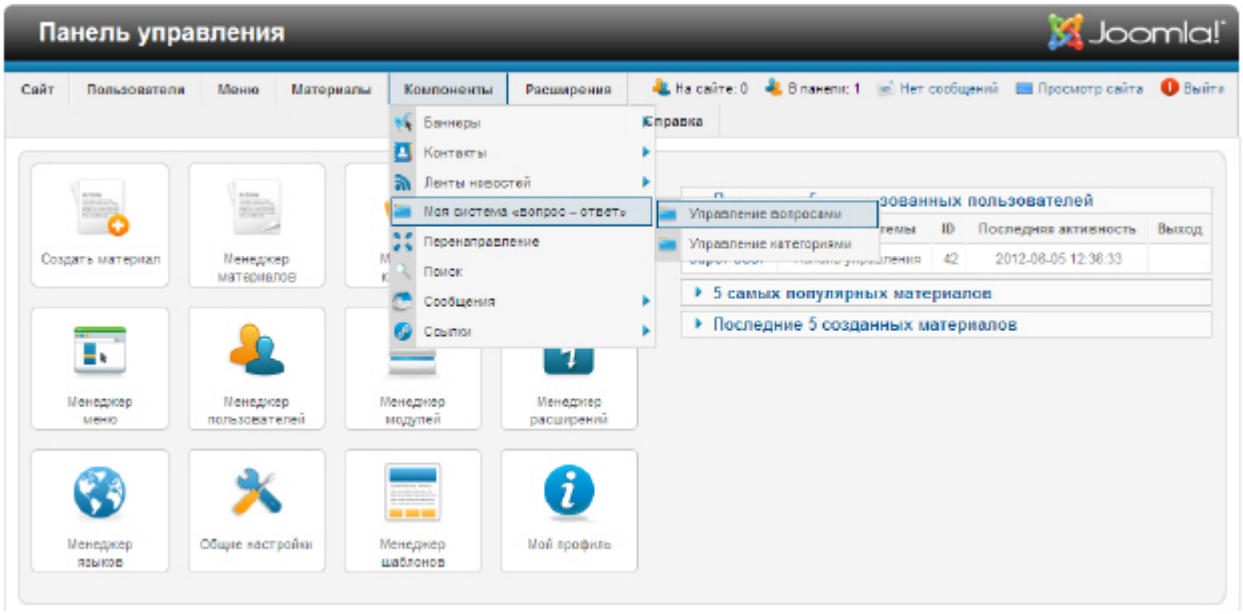


Рис. 4.8. Подпункты меню в бэкенде

Разработка фронтенда

Для удобства дальнейшего тестирования добавим еще несколько вопросов в различные категории:

```
INSERT INTO `jos_myquestions` (`id`, `name`, `date`, `question`, `city`,
`email`, `IP`, `id_cat`)
VALUES (NULL, 'Аноним', '2012-01-02 09:01:00', 'Быть или не быть?', 'Москва',
 'somebody@mail.ru', '12.345.67.890', '1');
INSERT INTO `jos_myquestions` (`id`, `name`, `date`, `question`, `city`,
`email`, `IP`, `id_cat`)
VALUES (NULL, 'Аноним', '2012-01-01 09:02:00', 'А судьи кто?', 'Москва',
 'somebody@mail.ru', '12.345.67.890', '2');
INSERT INTO `jos_myquestions` (`id`, `name`, `date`, `question`, `city`,
`email`, `IP`, `id_cat`)
VALUES (NULL, 'Аноним', '2012-01-01 09:03:00', ' А был ли мальчик?', 'Москва',
 'somebody@mail.ru', '12.345.67.890', '2');
```

Для наглядности выключите SEF в настройках сайта: в бэкенде зайдите в меню "**Сайт**" - "**Общие настройки**" и установите переключатель "**Включить SEF (ЧПУ)**" в значение "**Нет**".

Вывод списка категорий

Замените содержимое файла **/components/com_myquestions/myquestions.php** (обратите внимание, что мы больше не работаем с папкой **/administrator**) на следующий код:

```
<?php
defined('_JEXEC') or die('Restricted access');
$option = JRequest::getVar('option'); $task = JRequest::getVar('task');
jimport('joomla.application.helper');
require_once(JApplicationHelper::getPath('html'));
JTable::addIncludePath(JPATH_ADMINISTRATOR.DS.'components'.
DS.$option.DS.'tables');
switch($task)
{
  default:
    showCategories($option);
    break;
}
function showCategories($option)
{
  $db =& JFactory::getDbo();
  $query = "SELECT c.id, c.name, c.desc ".
        "FROM #__myquestions_categories c";
  $db->setQuery($query);
  $rows = $db->loadObjectlist();
  if ($db->getErrorNum())
  {
    echo $db->stderr();
    return false;
  }
  HTML_questions::showCategories($rows, $option);
}
?>
```

Мы подключаем файл **myquestions.html.php** с помощью require_once(JApplicationHelper::getPath('html')) и папку **table** бэкенда с помощью JTable::addIncludePath(). Обратите внимание, что мы обращаемся к классам таблиц бэкенда, хотя пишем фронтенд компонента.

Переключатель switch обрабатывает пока только значение задачи по умолчанию, вызывая функцию для получения списка категорий.

Теперь добавим класс HTML_questions для фронтенда. Создайте файл **/components/com_myquestions/myquestions.html.php**:

```
<?php
class HTML_questions
{
  function showCategories($rows, $option)
  {
    ?>
    <p><a href='index.php?option=<?=$option?>&task=showlist'><?=
     JText::_('COM_MYQUESTIONS_ALL_QUESTIONS')?></a></p>
    <table>
    <?php
```

```
    foreach($rows as $row)
    {
      $link = 'index.php?option='.$option.'&id_cat='.$row->id.'&task=showlist';
      echo '<tr><td><p><a href="' . $link . '">'.$row->name
      .'</a></td><td>'.$row->desc.'</td></tr>';
    }
    ?>
    </table>
    <?php
  }
}
?>
```

Класс HTML_questions будет содержать все функции вывода во фронтенде. Функция showCategories() принимает в качестве параметров массив записей таблицы базы данных и название текущего компонента. В цикле осуществляется обход этого массива и каждая запись выводится на экран.

Создайте файл **/language/ru-RU/ru-RU.com_myquestions.ini** и скопируйте в него строку:

```
COM_MYQUESTIONS_ALL_QUESTIONS="Все вопросы"
```

Теперь на странице компонента myquestions во фронтенде по умолчанию отображается список категорий (рис. 4.9).

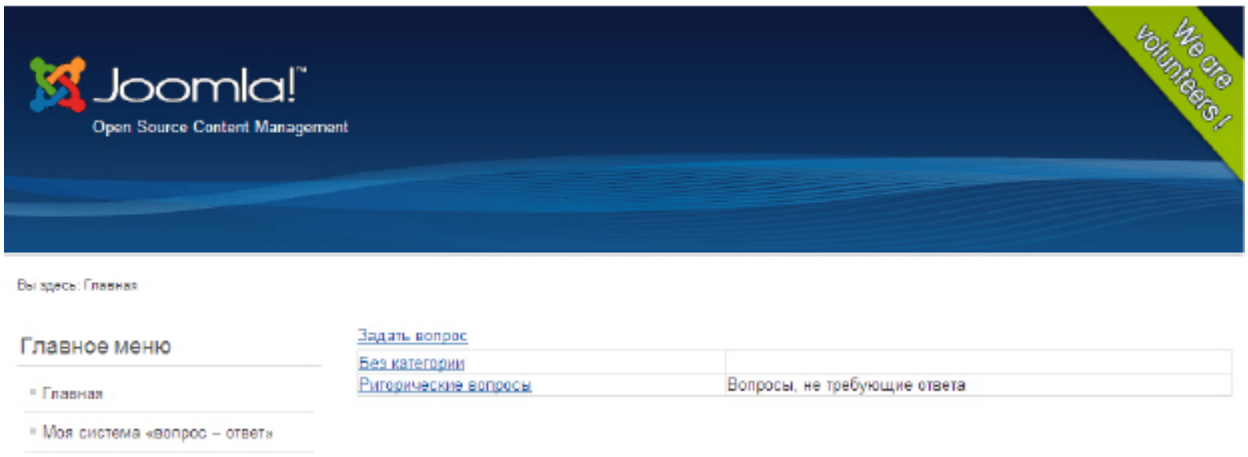


Рис. 4.9. Вывод списка категорий во фронтенде

Просмотр списка вопросов

Измените код конструкции switch в файле **myquestions.php**, добавив обработку задачи showlist:

```
case 'showlist':
  showQuestions($option);
  break;
```

Добавьте в этот же файл функцию showQuestions():

```
function showQuestions($option)
{
  $db =& JFactory::getDbo();
  $query = "SELECT
q.id,q.name,q.date,q.question,q.city,q.email,q.answer,q.id_cat,c.name AS cname
".
        "FROM #__myquestions q, #__myquestions_categories c ".
        "WHERE q.id_cat=c.id AND q.answer <> '' AND (q.published = 1 OR
        (q.expiration_date <> '0000-00-00 00:00:00' AND q.expiration_date >
NOW()))";

  $id_cat = JRequest::getVar('id_cat', '0');

  if ($id_cat != 0)
    $query .= "  AND q.id_cat = $id_cat";
  $db->setQuery($query);
  $rows = $db->loadObjectlist();
  if ($db->getErrorNum())
  {
    echo $db->stderr();
    return false;
  }

  if ($id_cat != 0)
  {
    $query = "SELECT name FROM #__myquestions_categories WHERE id=$id_cat";
    $db->setQuery($query);
    if ($db->getErrorNum())
    {
      echo $db->stderr();
      return false;
    }
    $name_cat = $db->loadResult();
  }
  else
    $name_cat = '';

  HTML_questions::showQuestions($rows, $option, $name_cat);
}
```

Длинный SQL-запрос требует пояснения. Как вы помните, при описании функциональности нашего компонента упоминалось, что отображаться на сайте будут вопросы, удовлетворяющие следующим условиям:

- есть ответ;
- либо вопрос не помечен как скрытый, либо дата снятия вопроса с публикации указана и больше текущей даты.

Данный запрос выбирает вопросы, которые соответствуют этим условиям.

Проверяется значение переменной $id_cat (id категории), и если категория задана, то из базы выбираются только вопросы, отнесенные к этой категории. Затем извлекается название категории для отображения над списком вопросов. Если же категория не задана, то выбираются вопросы из всех категорий.

Добавьте в класс HTML_questions в файле **myquestions.html.php** следующую функцию:

```
function showQuestions($rows, $option, $name_cat)
{
  if ($name_cat !== '')
    echo "<h1>$name_cat</h1>";

  foreach($rows as $row)
  {
    $link = 'index.php?option='.$option.'&id='.$row->id.'&task=showquestion';
    $link_cat = 'index.php?option='.$option.'&id_cat='.$row-
>id_cat.'&task=showlist';
    ?>
    <table width="100%">
      <tr>
        <td><i><?=$row->name?></i></td>
        <td><i><u><?=$row->email?></u></i></td>
        <td><i><?=JHTML::_('date', $row->date,
        JText::_('DATE_FORMAT_LC3'))?></i></td>
        <td><i><?=$row->city?></i></td>
      </tr>
      <tr>
        <td colspan="4"><a href="<?=$link_cat?>"><?=$row->cname?></a></td>
      </tr>
      <tr>
        <td colspan="4"><b><?=$row->question?></b></td>
      </tr>
      <tr>
        <td colspan="4"><?=$row->answer?></td>
      </tr>
      <tr>
        <td colspan="4"><a style="text-decoration: none;"
         title="<?=JText::_('COM_MYQUESTIONS_READMORE')?>" alt="<?=JText::_
         ('COM_MYQUESTIONS_READMORE')?>" href="<?=$link?>">---></a></td>
      </tr>
    </table>
    <br/>
    <?
  }
}
```

Данная функция выводит название категории, если оно задано, и затем проходит в цикле по всем записям, переданным ей. Для каждой записи выводится таблица с теми ее полями, которые были извлечены ранее.

Добавьте в файл **language/ru-RU/ru-RU.com_myquestions.ini** строку:

```
COM_MYQUESTIONS_READMORE="Подробнее"
```

Для проверки работоспособности нового кода нажмите на ссылку "**Все вопросы**" во фронтенде нашего компонента. Результат должен быть приблизительно таким, как на рис. 4.10.

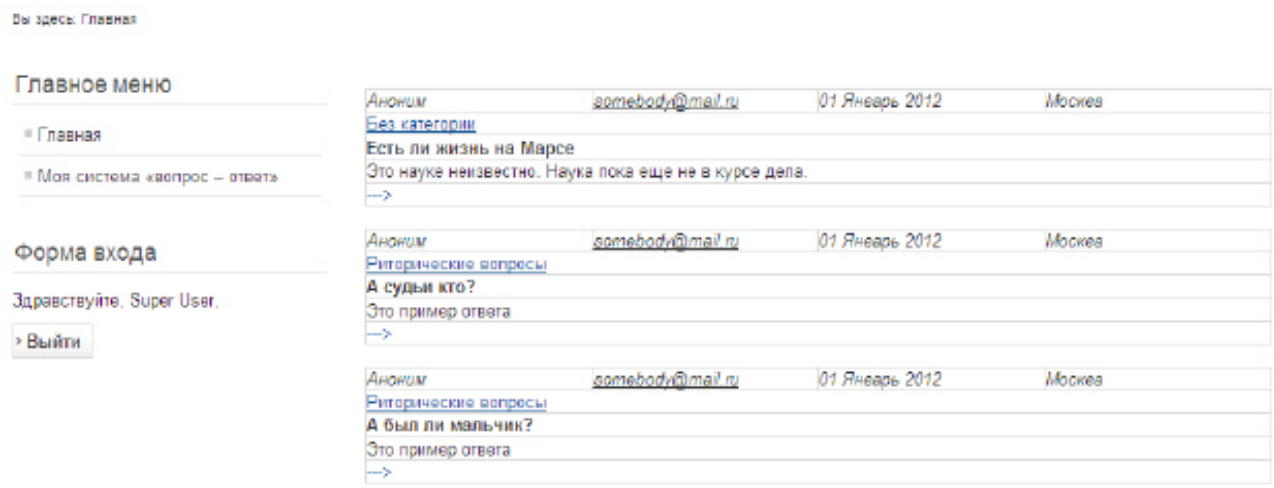


Рис. 4.10. Список вопросов во фронтенде

Вывод одного вопроса

Иногда может понадобиться вывести какой-либо один вопрос. Например, если пользователь захочет поделиться ссылкой на свой вопрос с кем-то другим.

Измените код конструкции switch в файле **myquestions.php**, добавив обработку задачи showquestion:

```
case 'showquestion':
  showQuestion($option);
  break;
```

Добавьте в этот же файл функцию showQuestion():

```
function showQuestion($option)
{
  $id = JRequest::getVar('id', 0) ;
  $row = &JTable::getInstance ('question', 'Table');
  $row->load($id);

  if ($row->answer == '' || ($row->published == 0 && ($row-
>expiration_date ==
   '0000-00-00 00:00:00' || strtotime($row->expiration_date) <=
time())))
    JError::raiseError(404, JText::_(' COM_MYQUESTIONS_ERROR404'));

  $row_cat =& JTable::getInstance('Category','Table');
  $row_cat->load($row->id_cat);

  HTML_questions::showQuestion($row, $option, $row_cat);
}
```

В данном коде мы получаем id из запроса к серверу с помощью функции getVar(). Если значение id отсутствует или неприемлемо, будет использовано значение по умолчанию, переданное во втором параметре (ноль). Затем мы получаем экземпляр класса таблицы из бэкенда и загружаем запись таблицы базы данных, соответствующую идентификатору id.

Далее происходит проверка того, можно ли отображать в открытом доступе данный вопрос. Исходя из условий отображения вопроса на сайте, нетрудно заметить, что не должны отображаться на сайте вопросы, удовлетворяющие хотя бы одному из условий:

- нет ответа;
- вопрос помечен как скрытый и дата снятия вопроса с публикации не указана или меньше или равна текущей дате.

Если вопрос не должен отображаться, то используется функция raiseError() класса JError для вывода сообщения "**404 - Вопрос не найден**". Такое же сообщение появится при попытке обращения к несуществующей записи.

Добавьте в файл **language/ru-RU/ru-RU.com_myquestions.ini** строку:

```
COM_MYQUESTIONS_ERROR404="Вопрос не найден"
```

Проверьте, что при обращении к вопросу, который не должен отображаться, выводится сообщение (рис. 4.11).

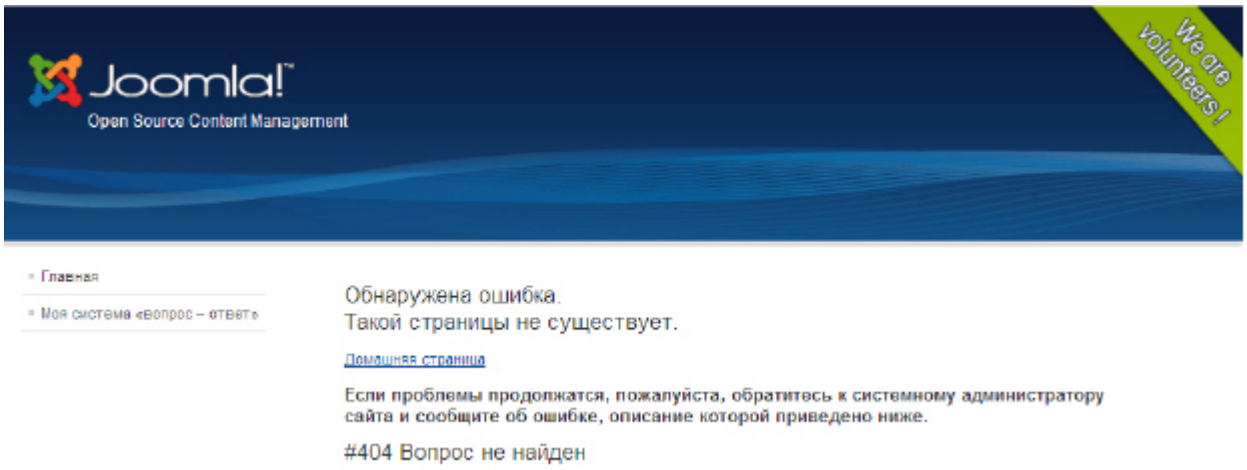


Рис. 4.11. Ошибка 404 "Вопрос не найден"

Теперь создадим функцию в классе вывода. Добавьте функцию showQuestion() в класс HTML_questions в файле **myquestions.html.php**:

```
function showQuestion($row, $option, $row_cat)
{
  $link_cat = 'index.php?option='.$option.'&id_cat='.$row-
>id_cat.'&task=showlist';
  ?>
  <p><a href='index.php?option=<?=$option?>&task=showlist'><?=JText::_
  ('COM_MYQUESTIONS_ALL_QUESTIONS')?></a></p>
  <table width="100%">
    <tr>
      <td><i><?=$row->name?></i></td>
      <td><i><u><?=$row->email?></u></i></td>
      <td><i><?=JHTML::_('date', $row->date,JText::_
      ('DATE_FORMAT_LC3'))?></i></td>
      <td><i><?=$row->city?></i></td>
    </tr>
    <tr>
      <td colspan="4"><a href="<?=
      $link_cat?>"><?=$row_cat->name?></a></td>
```

```
    </tr>
    <tr>
      <td colspan="4"><b><?=$row->question?></b></td>
    </tr>
    <tr>
      <td colspan="4"><?=$row->answer?></td>
    </tr>
  </table>
  <?
}
```

Данная функция аналогична функции showQuestions() за исключением того, что выводит не массив вопросов, а один вопрос.

Теперь в списке вопросов стрелка под каждым из них ("--->") ведет на страницу с выводом этого вопроса (рис. 4.12).

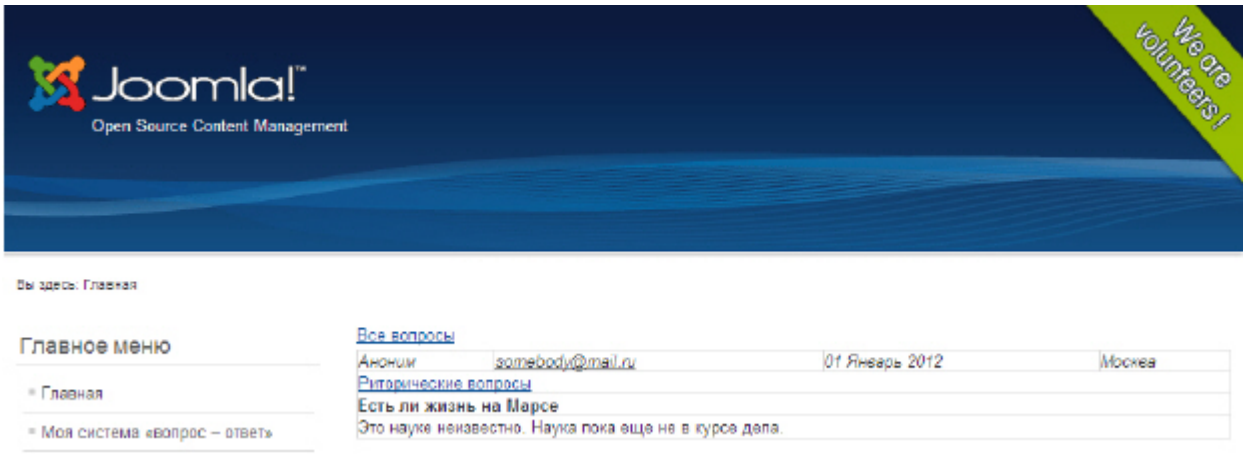


Рис. 4.12. Просмотр вопроса

Ключевые термины

JDate	- класс для работы с датами.
JEditor	- класс для работы с WYSIWYG-редактором.
JError	- класс для работы с ошибками.
JMail	- класс для создания и отправки электронных писем.
JMailHelper	- класс для очистки данных перед добавлением к электронному письму и проверки, является ли заданная строка корректным адресом электронной почты.
JURI	- класс для работы с URI.
Иерархия пунктов меню	- дерево, состоящее из пунктов меню и организованное с помощью вложенных множеств.

Краткие итоги

Для хранения иерархии пунктов меню в Joomla используются вложенные множества. Для каждого пункта в базе данных хранится id родителя, уровень и левый и правый ключи.

Создавать и отправлять электронные письма удобнее с помощью методов класса JMail.

Для работы с WYSIWYG-редактором используется класс JEditor.

Класс JURI позволяет работать с URI текущей или любой другой страницы.

Для работы с ошибками и предупреждениями может быть использован класс JError.

Для работы с датами в Joomla существует класс JDate.

Вопросы

1. Каким образом в базе данных Joomla хранится иерархия пунктов меню?
2. Какой класс позволяет создавать и отправлять электронные письма?
3. Каким образом можно отобразить код выбранного администратором сайта WYSIWYG-редактора?
4. Какой класс существует для работы с URI?
5. Для чего может быть использован класс JError?
6. Каким образом можно работать с датами в Joomla?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. список вариантов в дополнительных материалах).

5. Лекция: SEF-ссылки. Классы ядра JDocument, Juser

Рассмотрен процесс генерации и декодирования SEF-ссылок для какого-либо компонента. Рассмотрены классы для работы с документом и с данными текущего или любого другого пользователя.

Цель лекции: Изучить принципы работы с SEF-ссылками, сгенерированными в компоненте по собственному шаблону. Ознакомиться с некоторыми методами классов JDocument и JUser.

Генерация SEF-ссылок (класс JRoute)

В Joomla существует возможность создавать SEF-ссылки (Search-Engine Friendly) вида **www.mysite.ru/one/two/three**, удобные и для посетителей, и для поисковых систем. Для этого используется единственный метод класса JRoute, переводящий внутреннюю ссылку, генерируемую Joomla, в SEF-ссылку:

```
string _(string $url, bool $xhtml=true, int $ssl=null)
```

где

$url	- абсолютный или относительный URI;
$xhtml	- заменять ли амперсанды на "&";
$ssl	- при $ssl=1 полученный URI будет начинаться с протокола https://, в противном случае - http://.

Данный метод разбирает $url на пары "ключ-значение" и сохраняет результаты разбора в массиве. Из этого массива удаляется элемент option. Его значение добавляется к новому URL в качестве первого сегмента. Затем производится поиск файла **/components/com_<option>/router.php**. В этом файле должны находиться две функции: **функция для генерации SEF-ссылок** (<имя компонента>BuildRoute()) и **функция для декодирования элементов SEF-ссылок** (<имя компонента>ParseRoute()). Метод JRoute::_() вызовет функцию <имя компонента>BuildRoute() и передаст ей массив, полученный при разборе $url. Функция вернет массив $segments. Метод JRoute::_() добавит к новому URL все элементы этого массива, разделив их слэшами. Если в запросе останутся какие-либо необработанные переменные, они будут добавлены в конец URL.

Допустим, метод JRoute::_() получит на вход ссылку вида

```
index.php?option=com_mycomponent&var1=value1&var2=value2&…&varN=valueN
```

Тогда он передаст в функцию <имя компонента>BuildRoute() ассоциативный массив

```
Array([option]=>com_mycomponent [var1]=>value1 [var2]=>value2...
[varN]=>valueN)
```

причем пары "ключ-значение" будут расположены в том порядке, в котором они были в исходной ссылке. Задача функции <имя компонента>BuildRoute() - выбрать необходимые пары и сохранить в результирующем массиве только значения, но так, чтобы впоследствии можно было восстановить соответствующие им ключи. Это достигается использованием какого-либо фиксированного порядка. Функция вернет массив вида

```
Array([0]=>value1 [1]=>value2 … [N-1]=>valueN)
```

из которого затем будет создана SEF-ссылка

```
component/mycomponent/value1/value2/…/valueN
```

Впоследствии при щелчке на какой-либо SEF-ссылке произойдет обратный процесс. Будет вызван метод <имя компонента>ParseRoute(), который получит на вход массив

```
Array([0]=>value1 [1]=>value2 ... [N-1]=>valueN)
```

и вернет ассоциативный массив:

```
Array([var1]=>value1 [var2]=>value2 … [varN]=>valueN)
```

который Joomla установит в качестве переменных HTTP-запроса. Таким образом, посетители сайта и поисковые системы увидят SEF-ссылки, а компонент будет работать с обычным

набором пар "ключ-значение".

Обратите внимание, что SEF-ссылка не содержит никакой информации о том, как называются ключи массива. Чтобы их можно было восстановить, функция генерации ссылок и функция их декодирования должны неявно задавать **шаблон SEF-ссылок** для конкретного компонента. В данном примере подразумевается шаблон, который можно сформулировать вербально так: "переменные записываются в следующем порядке: var1, var2, … varN".

Шаблон может быть более сложным. Например, его структура может меняться в зависимости от задачи, т.е. от значения входящей в него переменной task.

В простейшем случае содержимое файла **router.php** выглядит так:

```
<?php
defined('_JEXEC') or die ('Restricted access');
function <имя компонента>BuildRoute(&$query)
{
  $segments = array();
  if (isset($query['var1']))
  {
    $segments[] = $query['var1'];
    unset($query['var1' ]) ;
  }
...
  if(isset($query['varN' ]))
  {
    $segments[] = $query['varN'];
    unset($query['varN']);
  }
  return $segments;
}
function <имя компонента>ParseRoute($segments)
{
  $vars = array();
  $vars['var1'] = @$segments[0];
...
  $vars['varN'] = @$segments[1];
  return $vars;
}
?>
```

Обратите внимание, что массив $query должен быть передан в функцию <имя компонента>BuildRoute() по ссылке. По мере заполнения массива $segments обработанные элементы удаляются из массива $query с помощью unset(). Любые элементы, которые останутся в массиве $query после работы функции <имя компонента>BuildRoute(), останутся и в URL. Если мы передадим $query по значению, вызовы функции unset() будут действовать только на локальную копию этого массива и все элементы старого URL будут появляться после SEF-сегментов.

Документ (класс JDocument)

Документ - это буфер, использующийся для хранения содержимого веб-страницы, которая будет показана пользователю после выполнения запроса.

Получение ссылки на глобальный объект JDocument:

```
$document =& JFactory::getDocument();
```

Этот объект хранит название, описание, язык, направление текста, дату модификации, кодировку и некоторые другие значения. Класс JDocument содержит несколько методов для получения этих значений: getTitle(), getDescription(), getLanguage(), getDirection(), getModifiedDate(), getCharset() и др. Соответственно, методы для задания этих значений называются setTitle(), setDescription() и т.д. и принимают в качестве аргумента новое значение.

Получение значения мета-тега

```
string getMetaData(string $name, bool $http_equiv = false)
```

где

$name	- название тега;
$http_equiv	- относится ли этот мета-тег к группе http-equiv (например, Content-Type, Refresh и др.).

Примеры:

```
echo $document->getMetaData('content-type', true);
echo $document->getMetaData('keywords', false);
```

Изменение значения мета-тега

```
void setMetaData(string $name, string $content, bool $http_equiv =
false, bool $sync = true)
```

где

$content	- значение атрибута content;
$sync	- синхронизировать ли тег content-type с MIME-типом документа.

Пример:

```
$document->setMetaData('content-type','text/html',true,true);
```

Добавление скриптов и каскадных таблиц стилей

Перечисленные ниже методы добавляют в секцию <head> соответствующие теги.

Добавление ссылки на скрипт:

```
void addScript(string $url, string $type = "text/javascript", bool
$defer = false, bool $async = false)
```

где

$url - URL скрипта;
$type - тип скрипта (text/javascript, text/vbscript и т.д.);
$defer - добавлять ли к тегу <script> атрибут defer="defer";
$async - добавлять ли к тегу <script> атрибут async="async".

Пример:

```
$document->addScript('/components/com_mycomponent/js/script.js');
```

Добавление непосредственно текста скрипта

```
void addScriptDeclaration(string $content, string $type =
'text/javascript')
```

где

$content - текст скрипта;
$type - тип скрипта.

Пример:

```
$document->addScriptDeclaration('alert("Hello World")');
```

Добавление внешней таблицы стилей

```
void addStyleSheet(string $url, string $type = 'text/css', string
$media = null, array $attribs = array())
```

где

$url - URL файла CSS;
$type - MIME-тип файла;
$media - значение атрибута media (screen, print, projection и др.);
$attribs - массив других атрибутов тега <link>.

Пример:

```
$document->addStyleSheet('/components/com_mycomponent/css/style.css');
```

Добавление внутренней таблицы стилей

```
void addStyleDeclaration(string $content, string $type = 'text/css')
```

где

$content - код CSS;
$type - значение атрибута type будущего тега <style>.

Пример:

```
$document->addStyleDeclaration('.myclass { color: red; }');
```

Пользователь (класс JUser)

Пользователь, просматривающий сайт, представлен объектом класса JUser, доступ к которому можно получить через метод getUser() класса JFactory:

```
$user =& JFactory::getUser();
```

Чтобы получить доступ к объекту, представляющему какого-либо другого пользователя, необходимо передать в метод getUser() id или логин этого пользователя:

```
$user =& JFactory::getUser(42);
$user =& JFactory::getUser('admin');
```

Поля класса JUser

JUser имеет ряд полей, для которых определен уровень доступа public и к которым поэтому можно обращаться непосредственно. Наиболее важные из них перечислены ниже (таблица 5.1).

Таблица 5.1. Некоторые public-поля класса JUser

Поле	Описание
block	Равно 1, если пользователь заблокирован
email	E-mail пользователя
guest	Равно 1, если пользователь является гостем, т.е. не залогинен
id	ID пользователя
lastvisitDate	Дата и время последнего входа пользователя в систему
name	Имя пользователя
params	Настройки пользователя
registerDate	Дата и время регистрации аккаунта пользователя
sendEmail	Равно 1, если пользователь согласен получать сообщения с сайта по электронной почте
username	Логин пользователя

Например, выведем приветствие для залогиненного пользователя:

```
if ($user->guest)
  echo "Пожалуйста, войдите в систему или зарегистрируйтесь";
else
  echo "Здравствуйте, {$user->name}! Последний раз вы были на сайте
".JHTML::_('date',$user->lastvisitDate);
```

Получение и изменение настроек пользователя

```
mixed getParam(string $key, mixed $default = null)
mixed setParam(string $key, mixed $value)
```

где

$key	- ключ параметра;
$default	- значение параметра по умолчанию;
$value	- устанавливаемое значение параметра.
setParam()	- возвращает предыдущее значение параметра.

Пример:

```
echo $user->getParam('language','ru-RU');
$user->setParam('language','en-GB');
```

Практика

Форма для написания вопроса

Измените код конструкции switch в файле **myquestions.php**, добавив обработку задачи showform:

```
case 'showform':
  showForm($option);
  break;
```

Добавьте в этот же файл функцию showForm():

```
function showForm($option)
{
  $user =&JFactory::getUser();
  if($user->name)
    $name = $user->name;
  else
    $name = '';

  HTML_questions::showForm($option, $name);
}
```

Перед вызовом функции вывода HTML-кода мы получаем имя залогиненного в настоящий

момент пользователя, если таковой имеется. Код $user = &JFactory::getUser() присваивает переменной $user ссылку на объект-представитель залогиненного пользователя. Если удалось получить имя пользователя, то мы сохраняем это значение в переменной $name, а в противном случае этой переменной присваивается пустая строка. Таким образом поле "**Автор**" в форме для написания вопроса будет уже заполнено, так что залогиненным пользователям не придется его заполнять.

Перейдите в файл **myquestions.html.php** и добавьте в класс HTML_questions метод showForm():

```
function showForm($option, $name)
{
  ?>
  <form action="index.php" method="post">
    <table>
      <tr>
        <td width="100">
            <?php echo JText::_('COM_MYQUESTIONS_AUTHOR');?>:
        </td>
        <td>
          <input class="text_area" type="text" name="name" id="name" size="50"
maxlength="255" value="<?php echo $name;?>"/>
        </td>
      </tr>
      <tr>
        <td width="100">
          <?php echo JText::_('COM_MYQUESTIONS_CITY');?>:
        </td>
        <td>
          <input class="text_area" type="text" name="city" id="city" size="50"
maxlength="50"/>
        </td>
      </tr>
      <tr>
        <td width="100">
          <?php echo JText::_('COM_MYQUESTIONS_EMAIL');?>:
        </td>
        <td>
          <input class="text_area" type="text" name="email" id="email" size="50"
maxlength="50"/>
        </td>
      </tr>
      <tr>
        <td width="100">
          <?php echo JText::_('COM_MYQUESTIONS_QUESTION');?>:
        </td>
        <td>
          <textarea name='question' id='question' class='inputbox' rows='15'
cols='38'></textarea>
        </td>
      </tr>
      <tr>
        <td width="100">
          <?php echo JText::_('COM_MYQUESTIONS_PUBLISHED');?>:
        </td>
        <td>
          <input type="hidden" name="published" value="0"/>
          <input type="checkbox" name="published" id="published" value="1"/>
        </td>
```

```
      </tr>
    </table>
    <input type="hidden" name="task" value="addquestion"/>
    <input type="hidden" name="option" value="<?php echo $option;?>"/>
    <input type="submit" class="button" id="button" value="<?php echo
JText::_('COM_MYQUESTIONS_SENDBUTTON');?>"/>
  </form>
  <?php
}
```

Листинг .

Как видите, в данной форме сразу два элемента с именем published. Дело в том, что, когда флажок установлен в выбранное состояние, то сценарию-обработчику формы в числе других параметров приходит пара "имя_флажка=значение". Однако когда флажок не установлен, эта пара не посылается. Поэтому используется следующий прием: перед флажком в форме помещается одноименное скрытое поле со значением, равным нулю. Тогда если флажок не установлен, то сценарий получит пару published=0. Если же он установлен, то сценарий тоже получит эту пару, но сразу же последует пара published=1, которая перекроет значение скрытого поля.

Так как мы поместили на форму скрытый элемент task со значением addquestion, то она будет обработана при обработке задачи addquestion. Поэтому добавьте в конструкцию switch в файле **myquestions.php** следующий код:

```
case 'addquestion':
  addQuestion($option);
  break;
```

Добавьте в этот же файл функцию addQuestion():

```
function addQuestion($option)
{
  $row =& JTable::getInstance('question', 'Table');
  if (!$row->bind(JRequest::get('post')))
  {
    echo "<script> alert('".$row->getError()."'); window.history.go(-1);
</script>\n";
    exit();
  }
  $row->question = nl2br(htmlspecialchars(JRequest::getVar('question', '',
'post', 'string',JREQUEST_ALLOWRAW), ENT_QUOTES));
  $row->IP = getenv('REMOTE_ADDR');
  $row->date = &JFactory::getDate()->toFormat();
  $row->id_cat = 1;

  if (!$row->store())
  {
    echo "<script> alert('".$row->getError()."'); window.history.go(-1);
</script>\n";
    exit();
  }

  $mailer =& JFactory::getMailer();
  $mailer->setSender('test@mysite.ru');
  $mailer->addRecipient('admin@mysite.ru');
  $mailer->setSubject(JText::_('COM_MYQUESTIONS_ADMIN_LETTER_SUBJECT'));
```

```
  $mailer->setBody(JText::sprintf('COM_MYQUESTIONS_ADMIN_LETTER_NEW_QUESTION',
$row->question));
  $mailer->IsHTML(true);
  if ($mailer->Send() !== true)
  {
    echo "<script> alert('".JText::_('COM_MYQUESTIONS_ADMIN_LETTER_ERROR')."');
window.history.go(-1); </script>\n";
    exit();
  }

  global $app;
  $app-> redirect(JRoute::_('index.php?option='.$option.'&task=view&view=all'),
    JText::sprintf('COM_MYQUESTIONS_QUESTION_SENT',$row->name));
}
```

Текст вопроса, введенный пользователем, пропускается через функцию htmlspecialchars(), преобразующую специальные символы в HTML-сущности. Таким путем предотвращается ввод нежелательных HTML-тегов. Затем результат пропускается через функцию nl2br(), вставляющую код разрыва строки
 перед каждым переводом строки, чтобы текст вопроса при выводе на веб-странице не слился в одну строку.

IP-адрес пользователя определяется с помощью функции getenv(), которая возвращает значение переменной окружения, в данном случае - REMOTE_ADDR.

По умолчанию вопросу присваивается категория с id, равным 1, то есть "Без категории".

Добавьте в файл **language/ru-RU/ru-RU.com_myquestions.ini** код:

```
COM_MYQUESTIONS_ADD_QUESTION="Задать вопрос"
COM_MYQUESTIONS_AUTHOR="Автор"
COM_MYQUESTIONS_DATE="Дата вопроса"
COM_MYQUESTIONS_QUESTION="Текст вопроса"
COM_MYQUESTIONS_CITY="Город"
COM_MYQUESTIONS_EMAIL="e-mail"
COM_MYQUESTIONS_CATEGORY="Категория"
COM_MYQUESTIONS_PUBLISHED="Отображать ли вопрос на сайте"
COM_MYQUESTIONS_SENDBUTTON="Отправить вопрос"
COM_MYQUESTIONS_QUESTION_SENT="Спасибо, %s! Ваш вопрос отправлен. Он будет
опубликован на сайте после получения ответа"
COM_MYQUESTIONS_ADMIN_LETTER_SUBJECT="Новый вопрос на сайте"
COM_MYQUESTIONS_ADMIN_LETTER_NEW_QUESTION="<p>Добрый день!</p><p>На сайте
появился новый вопрос:</p><p><i>%s</i></p>"
COM_MYQUESTIONS_ADMIN_LETTER_ERROR="Ошибка отправки письма"
```

Осталось добавить ссылку для написания вопроса. Измените начало функции showCategories() так:

```
function showCategories($rows, $option)
  {
    ?>
    <p><a href='index.php?option=<?=$option?>&task=showlist'><?
=JText::_('COM_MYQUESTIONS_ALL_QUESTIONS')
    ?></a></p>
    <p><a href='index.php?option=<?=$option?>&task=showform'><?
=JText::_('COM_MYQUESTIONS_ADD_QUESTION')
    ?></a></p>
```

Теперь на главной странице компонента во фронтенде появилась ссылка "**Задать вопрос**" (рис. 5.1).

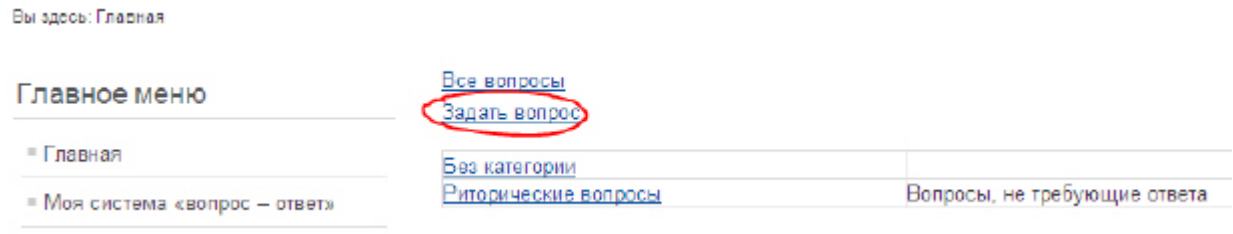


Рис. 5.1. Ссылка "Задать вопрос"

При переходе по этой ссылке появляется форма для написания вопроса (рис. 5.2). Обратите внимание, что в поле "**Автор**" подставилось имя текущего пользователя, если он залогинен.

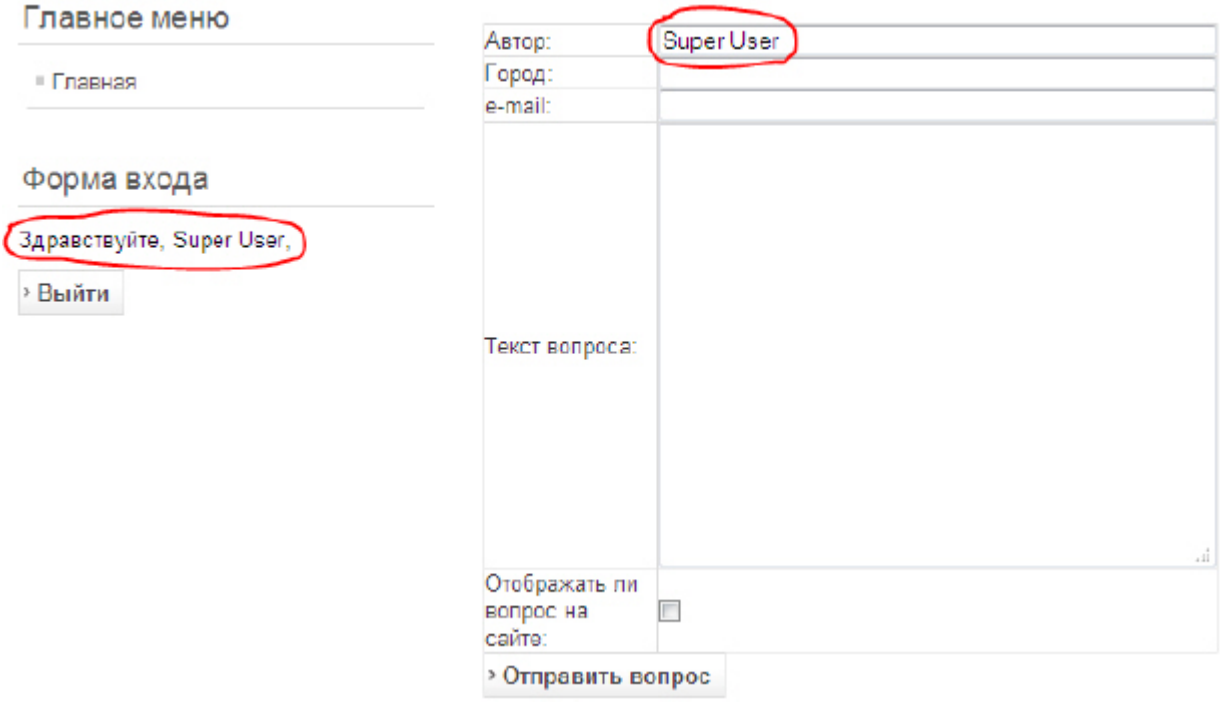


Рис. 5.2. Форма для написания вопроса

После написания вопроса и нажатия кнопки "**Отправить вопрос**" происходит перенаправление на главную страницу компонента с сообщением об успешной отправке вопроса (рис. 5.3).

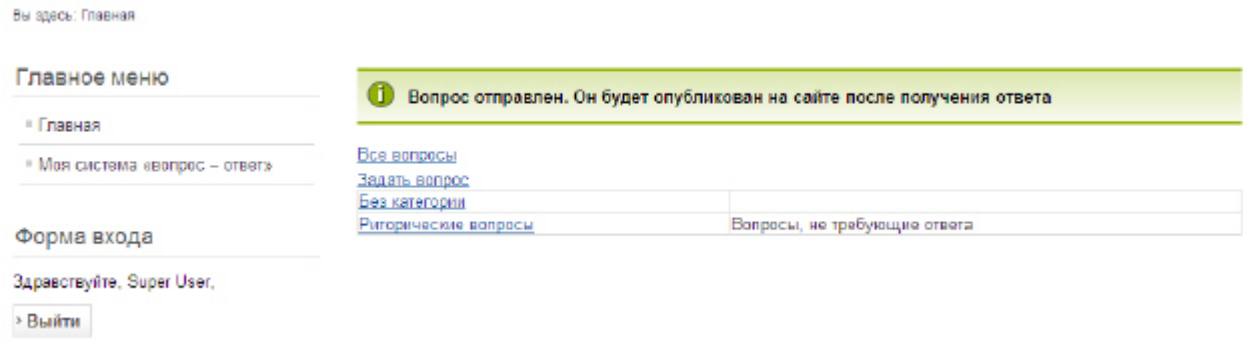


Рис. 5.3. Сообщение об отправке вопроса

Зайдите в папку **<путь к Денверу>/tmp/!sendmail** и найдите в ней файл ***.eml**, содержащий письмо-уведомление администратора о новом вопросе.

SEF

Включите SEF в бэкенде. Для этого перейдите в меню "**Сайт**" - "**Общие настройки**" и убедитесь, что переключатель "**Включить SEF (ЧПУ)**" установлен в "**Да**". Если вы используете в качестве веб-сервера Apache со включенным **mod_rewrite**, то вы можете также установить переключатель "**Перенаправление URL**" в "**Да**"; тогда из ваших ссылок исчезнет строка "**index.php**". Вид раздела "**Настройки SEO**" при включенном **mod_rewrite** показан на рис. 5.4.

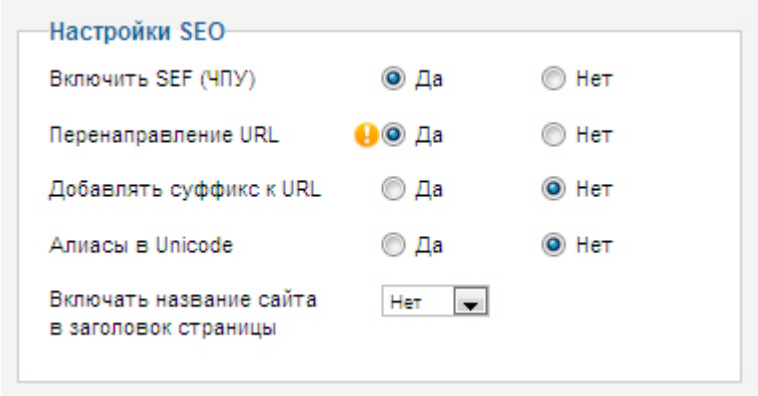


Рис. 5.4. Раздел панели управления "Настройки SEO"

Если ваша конфигурация не позволяет использовать mod_rewrite, SEF-ссылки все равно могут быть построены, но они будут включать строку "**index.php**", например: http://www.mysite.ru/index.php/one/two/three.

Нажмите кнопку "**Сохранить и закрыть**" для сохранения конфигурации. Если вы используете mod_rewrite, убедитесь, что вы переименовали находящийся в корневой папке Joomla файл **htaccess.txt** в **.htaccess** (если переименовать файл в проводнике Windows не удается, воспользуйтесь интерфейсом командной строки или каким-либо файловым менеджером, например, Total Commander).

Если вы получили сообщение о том, что ваша конфигурация не может быть перезаписана, задайте те же два значения вручную. Для этого откройте файл **configuration.php** в корневой папке Joomla, найдите строки:

```
public $sef = '0';
public $sef_rewrite = '0';
```

и измените оба значения на "**1" вместо "0**".

Генерация SEF-ссылок

Напишем функцию для генерации SEF-ссылок. Создайте файл **/components/com_myquestions/router.php**:

```
<?php
```

```
defined('_JEXEC') or die ('Restricted access');
function MyQuestionsBuildRoute(&$query)
{
  $segments = array();
  if (isset($query['task']))
  {
    $segments[] = $query['task'];
    unset($query['task' ]) ;
  }
  if(isset($query['id' ]))
  {
    $segments[] = $query['id'];
    unset($query['id']);
  }
  return $segments;
}
?>
```

Мы создаем пустой массив $segments. Затем проверяем, есть ли в массиве запроса элемент "task", и в этом случае добавляем значение задачи в массив $segments в качестве первого элемента и затем удаляем task из запроса. Далее мы повторяем тот же процесс для id. Наконец, возвращаем массив $segments, чтобы JRoute::_() могла закончить построение URL.

Исправим функции вывода нашего компонента так, чтобы они выводили SEF-ссылки вместо обычных. Откройте файл **/components/com_myquestions/myquestions.html.php** и измените код функции showCategories() класса HTML_questions следующим образом:

```
function showCategories($rows, $option)
{
  ?>
  <p><a href='<?=JRoute::_('index.php?option='.$option.'&task=showlist')?>'>
  <?=JText::_('COM_MYQUESTIONS_ALL_QUESTIONS')?></a></p>
  <p><a href='<?=JRoute::_('index.php?option='.$option.'&task=showform')?>'>
  <?=JText::_('COM_MYQUESTIONS_ADD_QUESTION')?></a></p>
  <table>
  <?php
  foreach($rows as $row)
  {
    $link = JRoute::_('index.php?option='.$option.'&id='.$row-
>id.'&task=showlist');
    echo '<tr><td><p><a href="' . $link . '">'.$row->name.
    '</a></td><td>'.$row->desc.'</td></tr>';
  }
  ?>
  </table>
  <?php
}
```

Измените выделенный код в функции HTML_questions::showQuestions():

```
foreach($rows as $row)
  {
    $link = JRoute::_('index.php?option='.$option.'&id='.$row-
>id.'&task=showquestion');
    $link_cat = JRoute::_('index.php?option='.$option.'&id_cat='.$row-
>id_cat.'&task=showlist');
    ?>
```

Измените также выделенный код в функции HTML_questions::showQuestion():

```
function showQuestion($row, $option, $row_cat)
{
  $link_cat = JRoute::_('index.php?option='.$option.'&id_cat='.$row-
>id_cat.'&task=showlist');
```

Теперь компонент будет генерировать SEF-ссылки по шаблону, установленному в функции MyQuestionsBuildRoute().

Декодирование SEF-ссылок

Если вы сейчас попытаетесь щелкнуть на одной из SEF-ссылок, то получите сообщение:

```
"Fatal error: Call to undefined function myquestionsParseRoute() in
Y:\home\localhost\www\joomla\includes\router.php on line …".
```

Напишем функцию для декодирования SEF-ссылок.

Откройте файл **/components/com_myquestions/router.php** и добавьте следующую функцию:

```
function MyQuestionsParseRoute ($segments)
{
  $vars = array();
  $vars['task'] = @$segments[0];
  $vars['id'] = @$segments[1];
  return $vars;
}
```

Как видите, в функции MyQuestionsParseRoute() мы считали переменные task и id из массива $segments в том же порядке, в котором мы их записывали в одноименный массив в функции MyQuestionsBuildRoute().

Знаки "@" при получении элементов массива $segments используются для подавления вывода сообщений об обращении к несуществующим элементам массива, т.к. не все наши SEF-ссылки будут содержать id.

Теперь щелкните по какой-либо ссылке во фронтенде и обратите внимание на строку статуса в браузере. Вы должны увидеть URL вида:
http://localhost/joomla/component/myquestions/showlist или
http://localhost/joomla/component/myquestions/showquestion/1

Ключевые термины

JDocument	- класс для работы с документом.
JRoute	- класс для создания SEF-ссылок.
JUser	- класс для работы с данными о пользователе.Документ
Документ	- буфер, использующийся для хранения содержимого веб-страницы, которая будет показана пользователю после выполнения запроса.
Функция генерации	- функция, которая принимает массив элементов HTTP-запроса и

SEF-ссылок	возвращает массив сегментов SEF-ссылки.
Функция декодирования SEF-ссылок	- функция, которая из массива сегментов SEF-ссылки создает массив переменных HTTP-запроса.
Шаблон SEF-ссылок	- последовательность сегментов.

Краткие итоги

SEF-ссылки в Joomla создаются с помощью метода JRoute::_(), который переводит внутреннюю ссылку, генерируемую Joomla, в SEF-ссылку. Чтобы компонент работал с SEF-ссылками, сгенерированными по собственному шаблону, необходимо создать в корневой папке его фронтенда файл **router.php**, в котором должны находиться функция для генерации SEF-ссылок и функция для их декодирования. Эти функции осуществляют взаимно обратные операции: первая из них из массива элементов HTTP-запроса создает массив сегментов SEF-ссылки, а вторая из массива сегментов SEF-ссылки создает массив переменных HTTP-запроса.

Так как SEF-ссылки не позволяют задать названия переменных запроса, то единственный способ определить, к какой переменной относится то или иное значение сегмента, - это использовать шаблон, который задает последовательность сегментов. Шаблон неявно задается в коде каждой из функций в файле **router.php**.

Для работы с документом и с данными пользователя в Joomla существуют соответственно классы JDocument и JUser.

Вопросы

1. Какой метод переводит внутреннюю ссылку, генерируемую Joomla, в SEF-ссылку?
2. Каким образом компоненты работают с SEF-ссылками?
3. Для чего служат функции генерации и декодирования SEF-ссылок?
4. Что такое шаблон SEF-ссылок и как он задается?
5. Какие классы используются для работы с документом и с данными пользователя?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. список вариантов в дополнительных материалах).

6. Лекция: Архитектура MVC в компонентах Joomla

Рассмотрены принципы реализации архитектуры MVC в компоненте и классы Joomla, использующиеся для этого.

Цель лекции: Ознакомиться с основами применения архитектуры MVC при разработке компонентов.

Взаимодействие элементов архитектуры MVC в Joomla

MVC ("Model - View - Controller") - это набор паттернов проектирования, который предполагает разделение программного кода на три группы:

- **модели** (model) используются для хранения данных. В Joomla модели реализуются с помощью абстрактного класса JModel;
- **представления** (view) генерируют вывод для заданной информации с помощью шаблона. В Joomla реализуются с помощью абстрактного класса JView;
- **контроллеры** (controller) получают команды от пользователя и управляют моделями и представлениями для выполнения этих команд. В Joomla реализуются с помощью абстрактного класса JController.

Приблизительно схема взаимодействия этих групп в коде Joomla представлена на следующей диаграмме последовательности (рис. 6.1 на основе иллюстрации из книги [4, p.246]).

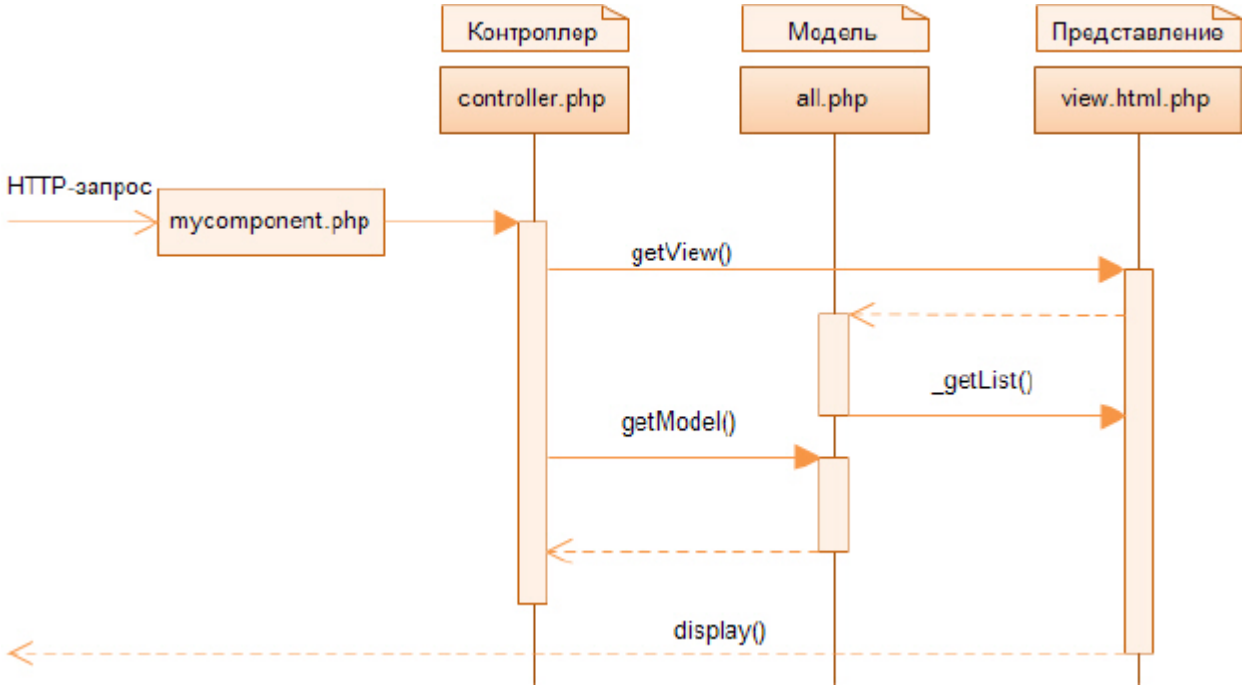


Рис. 6.1. Взаимодействие контроллера, модели и представления

В файле **/components/com_<имя компонента>/<имя компонента>.php** находится код для создания контроллера, например:

```
$controller = new MyComponentController();
$controller->execute(JRequest::getVar('task'));
$controller->redirect();
```

В HTTP-запросе задается задача, представление и, при необходимости, другие данные. Метод execute() вызывает метод вашего контроллера, который называется так же, как и заданная задача. Если задача не указана, то ей будет присвоено значение "display", следовательно, будет выполнен метод display(). Для этой и всех остальных задач, которые должен выполнять

ваш компонент, необходимо создать в классе контроллера одноименные методы. Наконец, метод redirect() перенаправляет пользователя к другому URL, если такой URL был задан в каком-либо методе при выполнении контроллера.

В простейшем случае класс контроллера описан в файле **/components/com_<имя компонента>/controller.php**, в более сложных случаях этих классов может быть несколько. Каждый из них должен быть производным от JController:

```
class MyComponentController extends JController
{
  …
  function display()
  {
    …
    parent::display();
        …
  }
…
}
```

Вы можете переопределить метод JController::display() в своем классе контроллера. Метод display() базового класса вызывает методы getView(), getModel(), а также метод display() заданного представления. getView() возвращает объект-представитель заданного представления, getModel() - заданной модели. По умолчанию используются те представление и модель, название которых совпадает с именем контроллера.

Далее нас будет интересовать работа метода display() заданного представления.

Каждый класс представления описан в файле **/components/com_<имя компонента>/views/<имя представления>/view.html.php** и является производным от JView. В этом классе может быть перегружен метод display(), чтобы вызвать метод класса модели для загрузки данных:

```
class MyComponentViewMyView extends JView
{
  function display($tpl=null)
  {
    $model=&$this->getModel();
    $list=$model->getList();
    $this->assignRef('list', $list);
    parent::display($tpl);
  }
}
```

Каждый класс модели описан в файле **/components/com_<имя компонента>/models/<имя модели>.php** и является производным от JModel. В этом классе может находиться метод для загрузки данных из базы данных или другого источника:

```
class ModelMyComponentMyModel extends JModel
{
  var $_somelist = null;
  function getList()
  {
    if (!$this->_somelist)
    {
```

```
      $query = "SELECT * FROM #__mycomponent";
      $this->_somelist = $this->_getList($query, 0, 0);
    }
    return $this->_somelist;
  }
}
```

Итак, метод класса представления display() вызывает метод класса модели для загрузки данных и сохраняет результат в какой-либо переменной, которая затем с помощью метода JView::assignRef() связывается с текущим представлением. Наконец, вызывается метод базового класса JView::display(), который загружает файл заданного шаблона при помощи перехвата выходного потока.

Шаблон находится в папке **/components/com_<имя компонента>/views/<имя представления>/tmpl**. В его коде осуществляется вывод на экран переменных текущего представления. Например:

```
<table width="100%">
  <?php
    foreach($this->list as $l)
      echo '<tr><td>'.$l->data.'</td></tr>';
  ?>
</table>
```

Так выглядит простейший вариант взаимодействия моделей, представлений и контроллеров.

Классы Joomla для реализации MVC

JModel

Одно из полей класса JModel - объект-представитель базы данных $_db. Таким образом, для выполнения запросов к базе данных в методах производных от JModel классов нужно обращаться непосредственно к этому полю, не получая новой ссылки на глобальный объект JDatabase:

```
$this->_db->setQuery($query);
$this->_db->query();
```

Получение списка каких-либо объектов и количества записей

```
array _getList(string $query, int $limitstart=0, int $limit=0)
int _getListCount(string $query)
```

где

$query	- запрос к базе данных;
$limitstart	- смещение;
$limit	- количество записей.

Например:

```
$query = "SELECT * FROM #__mycomponent";
$list = $this->_getList($query, 0, 0);
$count = $this->_getListCount($query);
```

JView

Joomla поддерживает возможность добавления нескольких моделей к одному представлению. В таком случае ссылки на объекты-представители моделей будут храниться в поле _models объекта JView. Для добавления модели используется метод

```
JModel setModel(object &$model, bool $default = false)
```

где

$model	- имя модели (т.е. имя соответствующего класса);
$default	- назначить ли ее моделью по умолчанию.

Метод возвращает добавленную модель.

Получение ссылки на объект-представитель одной из добавленных к представлению моделей

```
JModel getModel(string $name = null)
```

где $name - имя модели.

Например, добавим в коде контроллера к представлению SomeView модели Model1 и Model2:

```
$view = &$this->getView('SomeView', 'html');
$view->setModel($this->getModel('Model1'), true);
$view->setModel($this->getModel('Model2'));
```

Получение данных из зарегистрированной модели или поля представления

```
mixed get(string $property, string $default = null)
```

где

$property	- название метода модели, который требуется вызвать, или поля представления. В первом случае будет вызван метод get<Название метода>() - обратите внимание на заглавную букву;
$default	- если данные должны быть получены из модели, то $default - имя модели. Если требуется получить значение поля, то $default - значение, которое будет возвращено, если такое поле отсутствует.

Например, если в модели, заданной для текущего представления по умолчанию, есть метод getValue(), то получить в классе представления возвращаемое им значение можно так:

```
$temp = &$this->get('value');
```

Связывание переменной с представлением

```
bool assignRef(string $key, mixed &$val)
```

где

$key	- имя поля объекта-представителя представления. Не может начинаться со знака подчеркивания;
$val	- значение поля.

Пример:

```
$view->assignRef('somevar', $someval);
```

Выполнение и отображение скрипта шаблона

```
void display(string $tpl = null)
string loadTemplate(string $tpl = null)
```

где $tpl - имя файла шаблона. Конкретное имя файла зависит от значений имени и расширения макета, заданных в классе, по умолчанию это соответственно default и php. Если вы хотите изменить эти значения, используйте методы setLayout() и setLayoutExt(). Будет произведен поиск файла <имя макета>_<$tpl>.<расширение макета> или при $tpl=null <имя макета>.<расширение макета>.

display() выводит на экран результат работы скрипта шаблона, а loadTemplate() только возвращает этот результат. При ошибке display() возвращает объект Exception.

Например:

```
echo $view->loadTemplate('mytpl');
```

отобразит результат выполнения скрипта **/components/com_<имя компонента>/views/<имя представления>/tmpl/default_mytpl.php**.

JController

Выполнение задачи путем вызова одноименного метода производного класса

```
mixed execute(string $task)
```

где $task - имя задачи. Если такой задачи не найдется, будет выполнена задача "__default".

Метод возвращает значение, возвращаемое вызванным методом, или false в случае ошибки.

Например, код

```
$controller->execute('addItem');
```

приведет к вызову метода addItem() контроллера $controller.

Регистрация задачи

Регистрация задачи - это ее сопоставление какому-либо методу класса, производного от JController.

```
JController registerTask(string $task, string $method)
```

где

$task	- задача;
$method	- имя метода.

Пример в коде контроллера:

```
$this->registerTask('save', 'saveItem');
```

Стандартная реализация метода display()

О методе display() говорилось выше.

```
JController display(bool $cachable = false, array $urlparams = false)
```

где

$cachable	- задает, кэшировать ли вывод представления;
$urlparams	- массив пар "имя-значение" для URL, использующихся при кэшировании.

Получение ссылки на текущее представление

```
JView getView(string $name = '', string $type = '', string $prefix =
'', array $config = array())
```

где

$name - имя представления. По умолчанию совпадает с именем контроллера;
$type - тип представления, который можно определить как $document->getType();
$prefix - префикс класса представления. По умолчанию <имя контроллера>View;
$config - массив параметров, которые будут переданы в конструктор представления, - имя представления, кодировка, путь к директории шаблонов и т.д.

Если класс <префикс><имя представления> не найдется в директориях, заданных по умолчанию, то будет произведен его поиск в файле **<имя представления>/view.<тип представления>.php**. Например, если в коде класса MyComponentController есть строка

```
$view = &$this->getView('Item', 'html');
```

то будет произведен поиск класса MyComponentViewItem в файле **/components/com_<имя компонента>/views/Item/view.html.php**.

Получение объекта-представителя модели

```
JModel getModel(string $name = '', string $prefix = '', array $config
= array())
```

где

$name - имя модели. По умолчанию совпадает с именем контроллера;
$prefix - префикс класса модели. По умолчанию <имя контроллера>Model;
$config - массив параметров, которые будут переданы в конструктор модели.

Задание параметров для будущего перенаправления

```
JController setRedirect(string $url, string $msg=null, string
$type=null)
```

где

$url - URL для перенаправления;
$msg - сообщение для пользователя;
$type - тип сообщения. По умолчанию - "message".

Например, в коде контроллера можно написать:

```
$this->setRedirect('index.php?option=com_mycomponent', 'Текст
сообщения', 'notice');
```

Перенаправление браузера

```
bool redirect()
```

Метод возвращает false, если URL для перенаправления не был задан заранее.

Практика

Модели

Модель для списка всех категорий

В папке **/components/com_myquestions** создайте папку **models**, а в ней - файл **all.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.model');
class ModelMyQuestionsAll extends JModel
{
  var $_categories = null;
  function getList()
  {
    if (!$this->_categories)
    {
      $query = "SELECT id, name, `desc` FROM #__myquestions_categories";
      $this->_categories = $this->_getList($query, 0, 0);
    }
    return $this->_categories;
  }
}
?>
```

Мы подключаем библиотеку моделей Joomla и объявляем класс ModelMyQuestionsAll как производный от класса JModel. В классе хранится список категорий _categories. Метод getList() проверяет, загружен ли список категорий. Если нет, то мы создаем запрос, чтобы выбрать из базы данных все категории вопросов, и получаем их с помощью метода _getList() класса JModel.

Модель для списка вопросов из какой-либо категории или из всех категорий

Создайте файл **/components/com_myquestions/models/category.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.model');
class ModelMyQuestionsCategory extends JModel
{
  var $_questions = null;
  var $_id = null;
  var $_name = null;
  function __construct()
  {
    parent::__construct();
    $id = JRequest::getVar('id','all');
    $this->_id = $id;
```

```
  }
  function getList()
  {
    if (!$this->_questions)
    {
      if ($this->isAllCat())
        $id_text = "";
      else
        $id_text = " id_cat={$this->_id} AND ";
      $query = "SELECT q.id, q.question, q.name, q.date, q.email, q.city,
q.answer, c.id AS id_cat,
      c.name AS name_cat FROM #__myquestions q, #__myquestions_categories c
WHERE $id_text answer <>
      '' AND (published = 1 OR (expiration_date <> '0000-00-00 00:00:00' AND
expiration_date > NOW()))
      AND q.id_cat=c.id";
      $this->_questions = $this->_getList($query, 0, 0);
    }
    return $this->_questions;
  }
  function getCatName()
  {
    if (!$this->_name)
    {
      if (!$this->isAllCat())
      {
        $query = "SELECT name FROM #__myquestions_categories WHERE id = '" .
$this->_id . "'";
        $this->_db->setQuery($query);
        $this->_name = $this->_db->loadResult();
      }
    }
    if (!$this->isAllCat())
      return $this->_name;
    else
      return JText::_('COM_MYQUESTIONS_ALL_QUESTIONS');
  }
  Function isAllCat()
  {
    if ($this->_id=='all')
      return true;
    return false;
  }
}
?>
```

Листинг .

В данном классе хранятся список вопросов _questions, id категории _id и название категории _name.

Конструктор класса вызывает конструктор родительского класса, затем получает из HTTP-запроса id категории и сохраняет его в поле _id. Если id категории не задан, то вместо него сохраняется значение all, т.к. в таком случае будут выводиться вопросы сразу из всех категорий.

Метод getCatName() возвращает либо название текущей категории, либо строку "Все вопросы", если категория не задана.

Метод isAllCat() возвращает true, если категория не задана, и false в противном случае.

Модель для одного вопроса

Создайте файл **/components/com_myquestions/models/question.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.model');
class ModelMyQuestionsQuestion extends JModel
{
  var $_question = null;
  var $_id = null;
  function __construct()
  {
    parent::__construct();
    $id = JRequest::getVar('id',0);
    $this->_id = $id;
  }
  function getQuestion()
  {
    if (!$this->_question)
    {
      $query = "SELECT q.id, q.question, q.name, q.date, q.email, q.city,
q.answer, q.published, q.expiration_date,
      c.id AS id_cat, c.name AS name_cat FROM #__myquestions q,
#__myquestions_categories c WHERE q.id_cat=c.id AND
      q.id = {$this->_id}";
      $this->_db->setQuery($query);
      $this->_question = $this->_db->loadObject();
      if ($this->_question->answer == '' || ($this->_question->published == 0 &&
($this->
      _question->expiration_date == '0000-00-00 00:00:00' || strtotime($this-
>_question->expiration_date) <= time())))
      {
        JError::raiseError(404, JText::_(' COM_MYQUESTIONS_ERROR404'));
      }
    }
    return $this->_question;
  }
}
?>
```

Функция getQuestion() загружает одну запись с заданным id. Если после загрузки вопроса оказывается, что он является неопубликованным, то генерируется сообщение об ошибке 404.

Представления

Создайте в папке **/components/com_myquestions** подпапку **views**, а в ней - папки **all**, **category** и **question**. В каждой из них создайте по папке для шаблонов под названием **tmpl**.

Получившееся дерево папок показано на рис. 6.2.

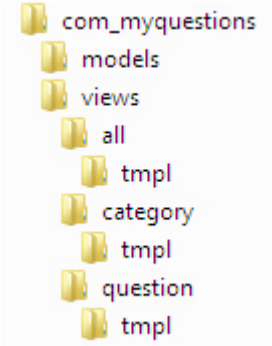


Рис. 6.2. Дерево папок MVC-компонента

Просмотр списка всех категорий

Создайте файл **/components/com_myquestions/views/all/view.html.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.view');
class QuestionViewAll extends JView
{
  function display($tpl=null)
  {
    global $option;
    $model=&$this->getModel();
    $list=$model->getList();
    for ($i=0; $i<count($list); $i++)
    {
      $row=&$list[$i];
      $row->link=JRoute::_('index.php?option='.$option.'&id='.$row-
>id.'&view=category&task=show');
    }
    $this->assignRef('list', $list);
    parent::display($tpl);
  }
}
?>
```

Класс QuestionViewAll объявляется как производный от класса JView.

В методе display() мы получаем ссылку на ассоциированную с данным представлением модель и используем ее метод getList(), чтобы получить список категорий. К каждому элементу этого списка добавляем ссылку для просмотра данной категории. Связываем этот список с переменной list шаблона и вызываем метод JView::display() для отображения шаблона.

По умолчанию будет отображен шаблон default. Напишем его. Создайте файл **/components/com_myquestions/views/all/tmpl/default.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
global $option;
echo "<a href=\"".JRoute::_('index.php?option='.
$option.'&view=question&task=showform')."\">".
JText::_('COM_MYQUESTIONS_ADD_QUESTION')."</a>";
?>
```

```
<br/>
<table width="100%">
  <?php
    foreach($this->list as $l)
      echo '<tr><td><p><a href="'.$l->link.'">'.$l->name.'</a></td><td>'.$l-
>desc.'</td></tr>';
  ?>
</table>
<br/>
```

Доступ к переменной list осуществляется через $this, т.к. это поле текущего класса QuestionViewAll.

Просмотр списка вопросов из какой-либо категории или из всех категорий

Создайте файл **/com_myquestions/views/category/view.html.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.view');
class QuestionViewCategory extends JView
{
  function display($tpl=null)
  {
    global $option;
    $model=&$this->getModel();
    $list=$model->getList();
    $name_cat=$model->getCatName();
    $is_all_cat=$model->isAllCat();
    for ($i=0; $i<count($list); $i++)
    {
      $row=&$list[$i];
      $row->link=JRoute::_('index.php?option='.$option.'&id='.$row-
>id.'&view=question&task=show');
      if ($is_all_cat)
        $row->link_cat=JRoute::_('index.php?option='.$option.'&id='.$row-
>id_cat.'&view=category&task=show');
    }
    $this->assignRef('list', $list);
    $this->assignRef('name_cat', $name_cat);
    $this->assignRef('is_all_cat', $is_all_cat);
    parent::display($tpl);
  }
}
?>
```

Данный код в целом аналогичен коду метода QuestionViewAll::display(). Если выводится список вопросов сразу из всех категорий, то в name_cat будет храниться текст "Все вопросы", а к объекту-представителю каждого вопроса добавится ссылка на его категорию. Если же выводится содержимое одной категории, то в name_cat будет храниться ее название, а ссылок на категорию каждого вопроса выводиться не будет, т.к. все эти ссылки будут одинаковы и вести на страницу с текущим же списком.

Для создания шаблона по умолчанию создайте файл **/components/com_myquestions/views/category/tmpl/default.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
global $option;
echo "<a href=\"".JRoute::_('index.php?option='.
$option.'&view=question&task=showform')."\">".JText::_('COM_MYQUESTIONS_ADD_QUES
TION')."</a>";
?>
<H1><?=$this->name_cat?></H1>
<?php foreach($this->list as $l): ?>
  <table width="100%">
    <tr>
      <td width="25%"><i><?=$l->name?></i></td>
      <td width="25%"><i><u><?=$l->email?></u></i></td>
      <td width="25%"><i><?=JHTML::_('date', $l->date,
      JText::_('DATE_FORMAT_LC3'))?></i></td>
      <td width="25%"><i><?=$l->city?></i></td>
    </tr>
    <?php
      if ($this->is_all_cat == true)
      {
    ?>
    <tr>
      <td colspan="4"><a href="<?=$l->link_cat?>"><?=$l->name_cat?></a></td>
    </tr>
    <?php
      }
    ?>
    <tr>
      <td colspan="4"><b><?=$l->question?></b></td>
    </tr>
    <tr>
      <td colspan="4"><?=$l->answer?></td>
    </tr>
    <tr>
      <td colspan="4"><a style="text-decoration: none;" title="<?
=JText::_('COM_MYQUESTIONS_READMORE')?>"
      alt="<?=JText::_('COM_MYQUESTIONS_READMORE')?>" href="<?=$l->link?>">---
></a></td>
    </tr>
  </table>
  <br/>
<?php endforeach;?>
```

Данный шаблон аналогичен шаблону по умолчанию для представления all.

Просмотр одного вопроса

Код для отображения одного вопроса аналогичен коду для отображения списка вопросов. Создайте файл **/components/com_myquestions/views/question/view.html.php**:

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.view');
class QuestionViewQuestion extends JView
{
  function display($tpl=null)
  {
if ($tpl !== 'form')
```

```
    {
      global $option;
      $model=&$this->getModel();
      $question=$model->getQuestion();
      $question->date=JHTML::Date($question->date);

      $this->assignRef('question', $question);
       $this->assignRef('option', $option);

      $this->assignRef('link_cat',JRoute::_('index.php?option='.$option.'&
      id='.$question->id_cat.'&view=category&task=show'));
    }
     parent::display($tpl);
  }
}
?>
```

Представление question будет соответствовать двум шаблонам - один для отображения вопроса, второй для вывода формы для отправки вопроса. Для первого шаблона необходимы данные о вопросе, которые мы получаем из модели. Для второго шаблона не требуется никаких данных кроме имени пользователя, которое мы определим в контроллере.

Напишем шаблон для отображения одного вопроса. Создайте файл **/components/com_myquestions/views/question/tmpl/default.php**:

```
<?php
  defined('_JEXEC') or die('Restricted access');
  global $option;
  echo "<a href=\"".JRoute::_('index.php?option='.
$option.'&view=question&task=showform')."\">"
.JText::_('COM_MYQUESTIONS_ADD_QUESTION')."</a>";
?>
<table width="100%">
  <tr>
    <td><i><?=$this->question->name?></i></td>
    <td><i><u><?=$this->question->email?></u></i></td>
    <td><i><?=JHTML::_('date', $this->question->date,
JText::_('DATE_FORMAT_LC3'))?></i></td>
    <td><i><?=$this->question->city?></i></td>
  </tr>
  <tr>
    <td colspan="4"><a href="<?=$this->
link_cat?>"><?=$this->question->name_cat?></a></td>
  </tr>
  <tr>
    <td colspan="4"><b><?=$this-
>question->question?></b></td>
  </tr>
  <tr>
    <td colspan="4"><?=$this->question->answer?></td>
  </tr>
</table>
```

Добавим другой шаблон, отображающий форму для написания вопроса. Создайте файл **/components/com_myquestions/views/question/tmpl/default_form.php**:

```
<?php
```

```
  defined('_JEXEC') or die('Restricted access');
?>
<form action="<?=JRoute::_('index.php')?>" method="post">
  <table>
    <tr>
      <td width="100">
         <?php echo JText::_('COM_MYQUESTIONS_AUTHOR');?>:
      </td>
      <td>
        <input class="text_area" type="text" name="name"
id="name" size="50" maxlength="255"
value="<?php echo $this->user_name;?>"/>
      </td>
    </tr>
    <tr>
      <td width="100">
        <?php echo JText::_('COM_MYQUESTIONS_CITY');?>:
      </td>
      <td>
        <input class="text_area" type="text"
name="city" id="city" size="50" maxlength="50"/>
      </td>
    </tr>
    <tr>
      <td width="100">
        <?php echo JText::_('COM_MYQUESTIONS_EMAIL');?>:
      </td>
      <td>
        <input class="text_area" type="text"
 name="email" id="email" size="50" maxlength="50"/>
      </td>
    </tr>
    <tr>
      <td width="100">
        <?php echo JText::_('COM_MYQUESTIONS_QUESTION');?>:
      </td>
      <td>
        <textarea name='question' id='question' class='inputbox' rows='15'
cols='38'></textarea>
      </td>
    </tr>
    <tr>
      <td width="100">
        <?php echo JText::_('COM_MYQUESTIONS_PUBLISHED');?>:
      </td>
      <td>
        <input type="hidden" name="published" value="0"/>
        <input type="checkbox" name="published" id="published" value="1"/>
      </td>
    </tr>
  </table>
  <input type="hidden" name="task"
value="addquestion"/>
  <input type="hidden" name="option"
value="<?=JRequest::getVar("option","")?>"/>
  <input type="submit" class="button" id="button"
 value="<?php echo JText::_('COM_MYQUESTIONS_SENDBUTTON');?>"/>
</form>
```

Листинг .

Создание контроллера

Создайте файл **/components/com_myquestions/controller.php** (метод addQuestion() скопируйте из файла **/components/com_myquestions/myquestions.php**, убрав параметр $option):

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.controller');
class QuestionController extends JController
{
  function display()
  {
    $document =& JFactory::getDocument();
    $viewName = JRequest::getVar('view', 'all');
    $viewType = $document->getType();
    $view = &$this->getView($viewName, $viewType);
    $model =& $this->getModel($viewName, 'ModelMyQuestions');
    if (!JError::isError($model))
    {
      $view->setModel($model, true);
    }
    $view->setLayout('default');
    $view->display();
  }
  function showForm()
  {
    $document =& JFactory::getDocument();
    $viewName = JRequest::getVar('view', 'question');
    $viewType = $document->getType();
    $view = &$this->getView($viewName, $viewType);
    $user =&JFactory::getUser();
    if($user->name)
      $view->user_name = $user->name;
    else
      $view->user_name = '';
    $view->display('form');
  }
  function addQuestion()
  {
    …
  }
}
?>
```

В методе display() мы получаем название запрашиваемого представления и тип текущего документа, который одновременно является и типом представления. Затем получаем ссылку на соответствующее представление и ссылку на одноименную модель. Добавляем модель к представлению, назначив ее по умолчанию. Задаем имя макета - default и вызываем метод JView::display(), который выполнит скрипт **/components/com_myquestions/views/all/tmpl/default.php**.

В методе showForm() мы также получаем объект-представитель текущего пользователя JFactory::getUser(), чтобы подставить его имя в форму для написания вопроса. Выражение $view->display('form') отображает шаблон из файла **default_form.php** (т.е. имя файла в

данном случае строится по схеме **"default"+"_"+tpl**, где tpl - параметр функции display()).

Метод addQuestion() добавляет новый вопрос в базу данных точно так же, как это делалось ранее. Обратите внимание на то, что название этого метода совпадает со значением, которое хранилось в скрытом элементе task формы для добавления вопроса:

```
<input type="hidden" name="task" value="addquestion"/>
```

Напишем код для создания объекта контроллера. Откройте файл **/components/com_myquestions/myquestions.php** и замените существующий код следующим:

```
<?php
defined('_JEXEC') or die('Restricted access');
require_once(JPATH_COMPONENT.DS.'controller.php');
JTable::addIncludePath(JPATH_ADMINISTRATOR.
DS.'components'.DS.'com_myquestions'.DS.'tables');
$controller = new QuestionController();
$controller->execute(JRequest::getVar('task'));
$controller->redirect();
?>
```

С помощью строки require_once(JPATH_COMPONENT.DS.'controller.php') подключается содержимое файла, содержащего код класса контроллера.

Изменение шаблона SEF-ссылок

Шаблон SEF-ссылок, использовавшийся нами до сих пор, не годится для применения в компоненте MVC, т.к. включает только переменные task и id. Для компонента MVC в URL должно быть задано еще по меньшей мере значение view.

Возможно, вы заметили, что в коде фронтенда, переделанном с учетом модели MVC, мы строили URL по шаблону **option/view/task/id** при включенных SEF и option=com_myquestions&view=value1&task=value2&id=value3 в противном случае. Для наглядности ниже приведено несколько примеров таких ссылок (таблица 6.1).

Таблица 6.1. Примеры ссылок, использующихся в MVC-версии компонента myquestions

Ссылка	view	task	id	Значение
/myquestions/category/show/1	category	show	1	Просмотр категории #1
/myquestions/question/show/1	question	show	1	Просмотр вопроса #1
/myquestions/category/show/all	category	show	all	Просмотр вопросов из всех категорий
/myquestions/all/show	all	show	-	Просмотр списка всех категорий
/myquestions/question/showform	question	showform	-	Вывод формы для написания вопроса

Изменим функции генерации и декодирования SEF-ссылок. Откройте файл **/components/com_myquestions/router.php** и измените код функции MyQuestionsBuildRoute() следующим образом:

```
function MyQuestionsBuildRoute(&$query)
{
  $segments = array();
    if (isset($query['view']))
```

```
  {
    $segments[] = $query['view'];
    unset($query['view']);
  }
  if (isset($query['task']))
  {
    $segments[] = $query['task'];
    unset($query['task']);
  }
  if (isset($query['id']))
  {
    $segments[] = $query['id'];
    unset($query['id']);
  }
  return $segments;
}
```

В том же файле замените функцию MyQuestionsParseRoute() следующей:

```
function MyQuestionsParseRoute ($segments)
{
  $vars = array();
  $vars['view'] = $segments[0];
  if (count($segments) > 1)
  {
        $vars['task'] = $segments[1];
    if (count($segments) > 2)
      $vars['id'] = $segments[2];
  }
  return $vars;
}
```

Как видите, теперь мы предполагаем, что первый элемент в массиве segments - это view, второй - task, а третий - id.

Добавление контроллера к коду бэкенда

Бэкенд не нуждается в большом контроле над форматом вывода, поэтому его можно не переводить на архитектуру MVC. Добавим только контроллер, чтобы исключить выражение switch().

Создайте файл **/administrator/components/com_myquestions/controller.php**. В нем мы объявим класс QuestionController. В конструкторе этого контроллера регистрируются задачи, взятые из старого кода переключателя switch из файла **admin.myquestions.php**.

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.application.component.controller');

class QuestionController extends JController
{
  function __construct($default = array())
  {
    parent::__construct($default);

    $this->registerTask('reply', 'replyToQuestion');
    $this->registerTask('save', 'saveQuestion');
```

```
    $this->registerTask('apply', 'saveQuestion');
    $this->registerTask('remove', 'removeQuestions');
    $this->registerTask('sendToExpert', 'send');
    $this->registerTask('sendAnswer', 'send');

    $this->registerTask('showCat', 'showCategories');
    $this->registerTask('addCat', 'editCategory');
    $this->registerTask('editCat', 'editCategory');
    $this->registerTask('saveCat', 'saveCategory');
    $this->registerTask('applyCat', 'saveCategory');
    $this->registerTask('removeCat', 'removeCategories');
  }
}
?>
```

Все функции из файла **admin.myquestions.php** перейдут в класс QuestionController в качестве методов практически без изменений, за исключением одного аспекта. Отказ от выражения switch ведет к невозможности передавать переменные непосредственно в методы класса контроллера. Поэтому необходимо либо добавлять в класс контроллера новые поля, либо получить переменные из переменных HTTP-запроса или других источников непосредственно в коде каждого метода. В нашем примере почти все методы используют значения переменных option и task. Теперь эти значения будут не передаваться как параметры, а извлекаться из HTTP-запроса с помощью функции JRequest(). Например, первые строки функции saveQuestion() примут вид:

```
function saveQuestion()
{
  $option = JRequest::getVar('option');
  $task = JRequest::getVar('task');
  $row = $this->save();
  ...
}
```

Итак, перенесите в класс QuestionController функции replyToQuestion(), save(), saveQuestion() и др. Затем замените содержимое файла **admin.myquestions.php** следующим кодом:

```
<?php defined('_JEXEC') or die('Restricted access');
require_once(JApplicationHelper::getPath('admin_html'));
require_once(JPATH_COMPONENT.DS.'controller.php');
JTable::addIncludePath(JPATH_COMPONENT.DS.'tables');
$controller = new QuestionController(array('default_task' => 'showQuestions'));
$controller->execute(JRequest::getVar('task'));
$controller->redirect(); ?>
```

Как вы уже заметили, конструктор нашего контроллера в бэкенде имеет параметр default. При вызове конструктора мы передаем в него массив, который хранит значение default_task, равное showQuestions. Таким путем задано название задачи, которая будет выполнена по умолчанию.

Ключевые термины

JController	- абстрактный класс для реализации контроллеров.
JModel	- абстрактный класс для реализации моделей.

JView	- абстрактный класс для реализации представлений.
Регистрация задачи	- сопоставление ее какому-либо методу класса, производного от JController.

Краткие итоги

Joomla поддерживает архитектуру MVC для компонентов. Модели, представления и контроллеры реализуются соответственно с помощью абстрактных классов JModel, JView и JController. В компоненте могут быть созданы классы, производные от всех или некоторых из этих классов.

Вот простейшая схема взаимодействия модели, представления и контроллера.

В файле, который находится в корневой папке компонента и называется так же, как компонент, находится код для создания контроллера и вызова его методов execute() и redirect(). Метод execute() вызывает метод контроллера, который называется так же, как и заданная задача.

Класс контроллера, производный от JController, содержит методы для каждой задачи, которую должен выполнять компонент. Метод JController::display(), который вызывается по умолчанию, вызывает методы getView(), getModel(), а также метод display() заданного представления.

В классе представления, производном от JView, может быть перегружен метод display() для вызова метода класса модели для загрузки данных и сохранения результата в какой-либо переменной. Затем с помощью метода JView::assignRef() эта переменная связывается с текущим представлением и вызывается метод базового класса JView::display(), который загружает файл заданного шаблона при помощи перехвата выходного потока.

В коде шаблона осуществляется вывод на экран переменных текущего представления.

Вопросы

1. Какие классы Joomla позволяют реализовать элементы архитектуры MVC?
2. Опишите схему взаимодействия модели, представления и контроллера.
3. Что такое регистрация задачи?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. список вариантов в дополнительных материалах).

7. Лекция: Модули. Постраничный вывод информации. Навигационная цепочка

Цель лекции: Понять принцип разработки модулей. Ознакомиться с методами классов JPagination и JPathway.

Модули

Как уже говорилось, модули в Joomla используются для отображения небольших фрагментов контента, обычно в левой или правой колонке или верхней или нижней областях страницы. Типичный модуль выводит информацию из таблицы какого-нибудь компонента, например, случайную фотографию или несколько последних статей.

По сравнению с разработкой компонента написать модуль значительно легче. Как правило, модуль не использует собственных таблиц и не обрабатывает данные, введенные пользователем. Код модуля может даже поместиться в одном-единственном файле.

Постраничный вывод информации (класс JPagination)

Joomla позволяет разбивать длинные списки на страницы, задавая длину списка по умолчанию (**"Сайт" - "Общие настройки", выпадающий список "Длина списка по умолчанию"**). Для вывода списков элементов с разбивкой на страницы как в бэкенде, и так и во фронтенде используется класс JPagination. Его открытые (public) поля хранят следующую информацию:

total	- общее количество записей;
limitstart	- порядковый номер записи, с которой нужно начать вывод;
limit	- количество записей на страницу;
prefix	- префикс переменных запроса.

Соответственно, конструктор принимает эти четыре значения в качестве параметров:

```
__construct(int $total, int $limitstart, int $limit, string $prefix = '')
```

Например:

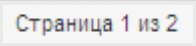
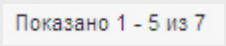
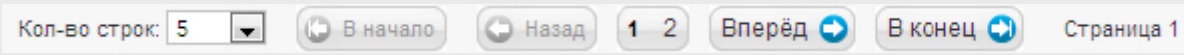
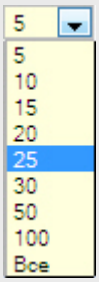
```
$paginationObject = new JPagination(100, 20, 10, 'somePrefix');
```

Как вы, возможно, помните, второй и третий параметры метода JDatabase::setQuery() - это смещение для начала выборки и количество выбираемых строк. Они в точности соответствуют параметрам limitstart и limit. Поэтому используйте одни и те же значения для создания объекта JPagination и для задания параметров setQuery():

```
$db->setQuery("SELECT * FROM #__mycomponent", $limitstart, $limit);
$rows = $db->loadObjectList();
jimport('joomla.html.pagination');
$paginationObject = new JPagination($total, $limitstart, $limit);
```

После создания объекта JPagination необходимо вызвать один из его методов для отображения счетчика страниц, ссылки на предыдущую/следующую страницу и т.д. Все эти методы не имеют параметров и возвращают HTML-код соответствующих элементов. Для наглядности посмотрите на результаты работы этих методов (таблица 7.1).

Таблица 7.1. Методы класса JPagination

Метод	Результат
getPagesCounter()	
	Рис. 7.1. Результат работы метода getPagesCounter()
getResultsCounter()	
	Рис. 7.2. Результат работы метода getResultCounter()
getPagesLinks()	
	Рис. 7.3. Результат работы метода getPagesLinks()
getListFooter()	
	Рис. 7.4. Результат работы метода getListFooter()
getLimitBox()	
	Рис. 7.5. Результат работы метода getLimitBox()

Например:

```
echo $paginationObject->getListFooter();
```

Также класс JPagination содержит методы orderUpIcon() и orderDownIcon() для вывода стрелок "вверх" и "вниз", использующихся для задания собственного порядка записей.

Управление навигационной цепочкой (класс JPathway)

Навигационная цепочка ("хлебные крошки") - это последовательность элементов, представляющая собой путь по сайту от корня до текущей страницы. Для управления навигационной цепочкой в Joomla существует объект **JPathway**, доступ к которому можно получить так:

```
global $app;
$pathway =& $app->getPathway();
```

Добавление элемента в навигационную цепочку

```
bool addItem(string $name, string $link='')
```

где

$name - название (текст) элемента;
$link - гиперссылка.

Например:

```
$pathway->addItem('Категория #1','mycomponent/category/1');
$pathway->addItem('Элемент #1');
```

Получившаяся цепочка показана на рис. 7.6.

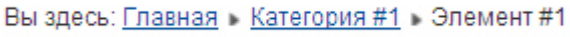


Рис. 7.6. Навигационная цепочка

Обратите внимание, что для элемента, который окажется в цепочке последним, ссылка выводиться не будет, даже если она задана. Это понятно, т.к. последний элемент соответствует текущей странице, ссылка на которую и без того известна. Тем не менее, такая ссылка не теряется и сохраняется в массиве _pathway, в котором класс JPathway хранит элементы цепочки как объекты с двумя полями - name и link.

Получение массива элементов навигационной цепочки

```
array getPathway()
```

Для приведенного выше примера массив выглядит так:

```
Array ([0]=>stdClass Object ([name]=>Категория #1
[link]=>mycomponent/category/1)
 [1]=>stdClass Object ([name]=>Элемент #1 [link]=>))
```

Получение только названий элементов без ссылок

```
array getPathwayNames()
```

Для того же примера этот метод вернет массив

```
Array ([0]=>[1]=>Категория #1 [2]=>Элемент #1)
```

Изменение названия заданного элемента

```
bool setItemName(int $id, string $name)
```

где

$id	- индекс элемента;
$name	- новое название.

Пример:

```
$pathway->setItemName(0,'Категория #2');
```

Вид навигационной цепочки после выполнения этого кода показан на рис. 7.7.

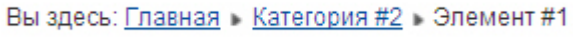


Рис. 7.7. Измененная навигационная цепочка

Замена массива элементов навигационной цепочки

```
array setPathway(array $pathway)
```

где $pathway - новый массив объектов цепочки (т.е. для каждого элемента должны быть заданы поля name и link).

Метод возвращает предыдущее значение массива.

Например:

```
$item1->name = "item1";
$item1->link = "link1";

$item2->name = "item2";
$item2->link = "link2";

$item3->name = "item3";
$item3->link = "";

$items = array($item1, $item2, $item3);

$pathway->setPathWay($items);
```

Получившаяся цепочка показана на рис. 7.8.

Рис. 7.8. Навигационная цепочка после замены массива ее элементов

Практика

Разработка модуля

Регистрация модуля в базе данных

Модуль, как и компонент, необходимо зарегистрировать в базе данных. Для этого создадим запись в таблицах #__modules и #__extensions. Выполните следующие SQL-запросы:

```
INSERT INTO jos_modules (title, ordering, position, published, module,
showtitle, params)
VALUES ('Новые вопросы', 1, 'position-7', 1,
 'mod_myquestions', 1, '{"random":"0","items":"3","maxlen":
"100","author":"1","date":"1"}');
INSERT INTO
jos_extensions(name,type,element,folder,client_id,manifest_cache,params,custom_d
ata,system_data)
VALUES('mod_myquestions', 'module', 'mod_myquestions', '', 0,
'{"legacy":false,"name":"mod_myquestions",
"type":"module","creationDate":"2012","author":"Me",
"copyright":"","authorEmail":"","authorUrl":"","version":"1.6.0",
"description":"My Questions module","group":""}', '{}', '', '');
```

Если вы обновите фронтенд после выполнения этих запросов, вы заметите, что модуль не появился, несмотря даже на то, что полю published присвоено значение 1. Дело в том, что модуль должен быть не только опубликован, но еще и назначен для каких-либо пунктов меню. Для этого в панели администрирования войдите в меню "**Расширения**" - "**Менеджер модулей**" и выберите модуль "**Вопросы**". В разделе "**Привязка к пунктам меню**" выберите в выпадающем списке значение "**На всех страницах**" и нажмите кнопку "**Сохранить и закрыть**" (рис. 7.9).

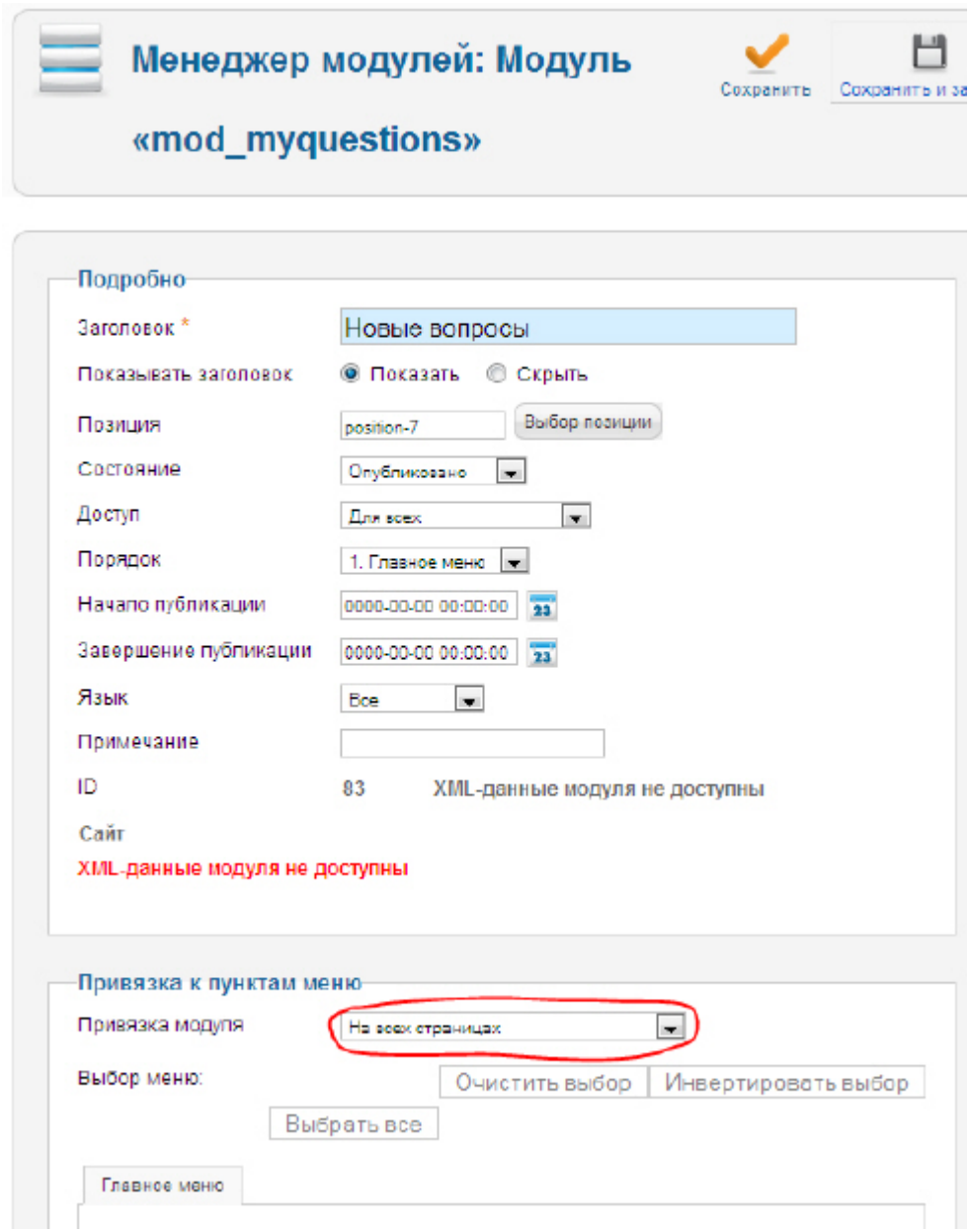


Рис. 7.9. Привязка модуля ко всем страницам

Создание модуля

Напишем модуль, который будет выводить ссылки на последние вопросы или на случайный вопрос. Создадим в папке **/modules** папку **mod_myquestions**, а в ней - файл **mod_myquestions.php**:

```
<?php
defined('_JEXEC') or die ('Restricted access');

$items = $params->get('items', 1);
$maxlen = $params->get('maxlen',100);
$random = $params->get('random', 0);
$q_author = ($params->get('author',1) == 1) ? ", name" : "";
$q_date = ($params->get('date',1) == 1) ? ", date" : "";

$db = &JFactory::getDbo();
```

```
$query = "SELECT id, question$q_author$q_date FROM #__myquestions WHERE answer
<> ''
AND (published = 1 OR (expiration_date <> '0000-00-00 00:00:00' AND
expiration_date > NOW()))";
if ($random)
{
  $orderby = " ORDER BY RAND()";
  $items = 1;
}
else
  $orderby = " ORDER BY date DESC";
$query .= $orderby;
$db->setQuery($query, 0, $items);
$rows = $db->loadObjectList();
foreach($rows as $row)
{
  echo '<a href="'.JRoute::_("index.php?
option=com_myquestions&view=question&task=show&
  id=".$row->id).'">'.substr(strip_tags($row->question),0,$maxlen-
1).'</a><br/>';
  $addition = array();
  if ($params->get('author',1) == 1)
    $addition[] = $row->name;
  if ($params->get('date',1) == 1)
    $addition[] = JHTML::_('date', $row->date, JText::_('DATE_FORMAT_LC3'));
  if (count($addition))
    echo '<i>'.implode(' ', $addition).'</i><br/>';
}
?>
```

Для установки и считывания параметров может быть использован глобальный объект $params. Когда мы добавили запись в таблицу #__modules, поле params содержало пять значений: random, равное 0, items, равное 3, maxlen, равное 100, author, равное 1, date, равное 1. Их значения мы получаем с помощью метода get(), второй параметр которого - это значение по умолчанию.

Далее задается SQL-запрос для получения из таблицы #__myquestions вопросов, подлежащих публикации. Значения id и question мы получаем в любом случае, а name и date - если в настройках модуля задано, что нужно выводить эти значения. Если нужно выводить один случайный вопрос, то к запросу добавляется "ORDER BY RAND()" и количество записей, которое мы хотим получить, вне зависимости от значения items, заданного в настройках, задается равным 1. В противном случае нужно выводить несколько самых новых вопросов, поэтому к запросу добавляется "ORDER BY date DESC" для сортировки по дате вопроса.

Второй и третий параметры функции setQuery() используются для задания оператора LIMIT, который автоматически добавится к запросу. Таким путем мы получаем из базы данных только первые items записей.

Для каждой полученной записи выводится ссылка на ее просмотр с помощью JRoute::_() и первые maxlen символов, а также, если это разрешено в настройках, имя автора и дата.

Теперь во фронтенде появится модуль, показанный на рис. 7.10. Если модуль не отображается, попробуйте изменить позицию с position-7 на какую-либо другую.

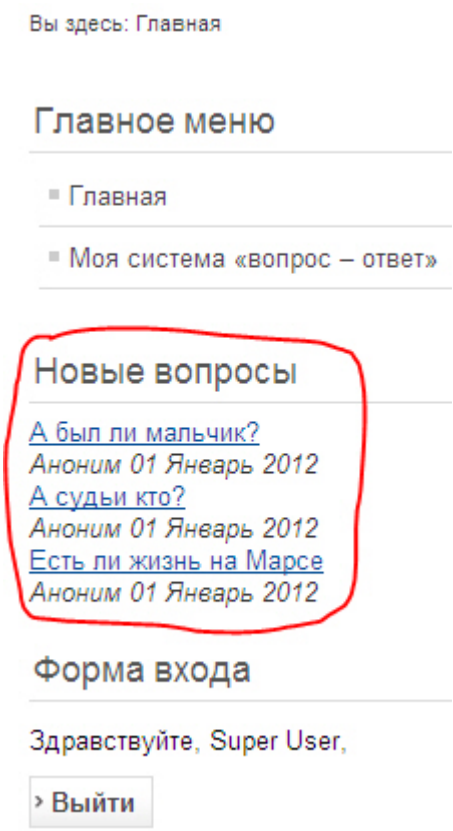


Рис. 7.10. Модуль, отображающий три самых новых вопроса

С помощью phpMyAdmin измените значение random в настройках модуля mod_myquestions (поле params таблицы #__modules) на "1":

```
{"random":"1","items":"3","maxlen":"100","author":"1","date":"1"}
```

Теперь модуль выводит один случайный вопрос (рис. 7.11).

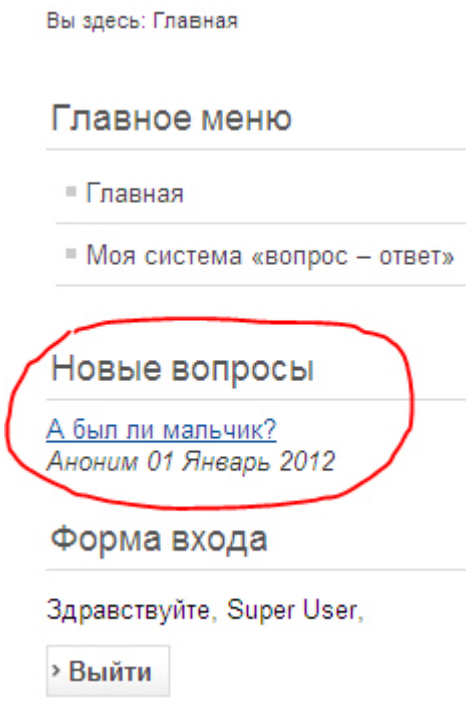


Рис. 7.11. Модуль, отображающий один случайный вопрос

Постраничный вывод информации

Добавим к менеджеру вопросов постраничный вывод списка. Откройте файл **/administrator/components/com_myquestions/controller.php** и сделайте изменения в коде функции showQuestions() в соответствии с выделенным кодом:

```
function showQuestions()
  {
    $option = JRequest::getVar('option');
    global $app;
    $limit = JRequest::getVar('limit', $app->getCfg('list_limit'));
    $limitstart = JRequest::getVar('limitstart', 0);

    $db =& JFactory::getDbo();
    $query = "SELECT count(*) FROM #__myquestions";
    $db->setQuery($query);
    $total = $db->loadResult();

    $query = "SELECT * FROM #__myquestions";
    $db->setQuery($query, $limitstart, $limit);
    $rows = $db->loadObjectList();
    if ($db->getErrorNum())
    {
      echo $db->stderr();
      return false;
    }
    jimport('joomla.html.pagination');
    $pageNav = new JPagination($total, $limitstart, $limit);
    HTML_questions::showQuestions($option, $rows, $pageNav);
  }
```

Значения limit и limitstart мы получаем из HTTP-запроса. Их задает один из методов класса JPagination, который мы вызовем далее. По умолчанию limit берется из настроек Joomla, а limitstart принимается равным 0. Значение total мы получаем из базы данных.

Когда все данные получены, создается объект JPagination, который передается в метод HTML_questions::showQuestions().

Откройте файл **admin.myquestions.html.php** и измените метод showQuestions() следующим образом:

```
function showQuestions($option, &$rows, &$pageNav)
{
  $maxlen = 100;
?>
  <form action="index.php" method="post"
  name="adminForm">
    <table class="adminlist">
    ...
    <?php
        jimport('joomla.filter.output');
        $k = 0;
        for  ($i = 0,  $n = count($rows); $i < $n;  $i ++)
        {
...
        }
    ?>
      <tfoot>
        <td colspan="10">
          <?php echo $pageNav->getListFooter();?>
        </td>
      </tfoot>
    </table>
    ...
  </form>
<?php
}
```

Код, полученный с помощью getListFooter(), выведет выпадающий список для выбора количества элементов на странице и ссылки на другие страницы. Название элемента <select> - limit, таким образом, выбранное в выпадающем списке значение попадет в HTTP-запрос как значение переменной limit. Также getListFooter() добавит к форме скрытое поле limitstart, значение которого будет изменяться в зависимости от limit и от нажатой пользователем ссылки на страницу: 0 для первой страницы, limit для второй, 2*limit для третьей и т.д. Таким образом переменная limitstart также будет включена в HTTP-запрос. Из него при переходе на другую страницу эти значения получит функция showQuestions() и использует их при задании нового запроса к базе данных.

Для проверки задайте в настройках сайта длину списка по умолчанию, равную 5, и добавьте вопросы так, чтобы их количество стало больше, чем 5. Примерный вид, который теперь примет список вопросов в бэкенде, показан на рис. 7.12.

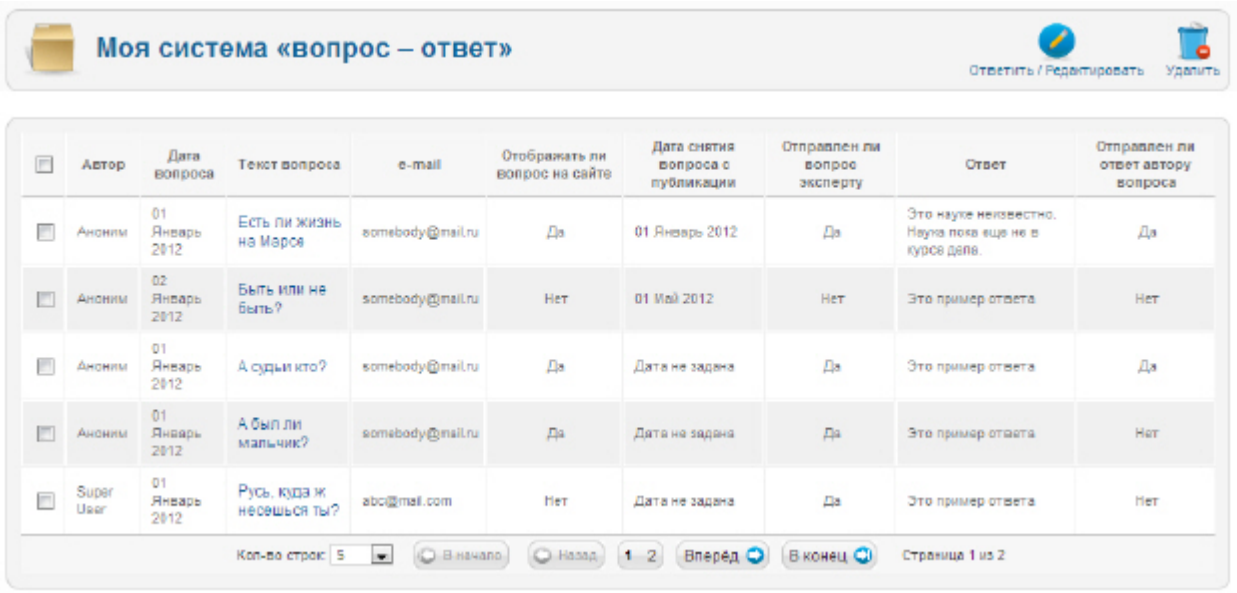


Рис. 7.12. Список вопросов с разбиением на страницы

Нацигационная цепочка

Рассмотрим, какие элементы можно отобразить в навигационной цепочке для каждого из представлений нашего компонента:

all		- название компонента;
category		- название компонента, название категории;
question	шаблон default	- название компонента, название категории, текст вопроса;
	шаблон default_form	- название компонента, текст "Задать вопрос".

Как видите, название компонента нужно выводить для всех представлений, поэтому его лучше вывести уже в методе контроллера. В методах представлений будем выводить только те элементы, которые требуют какой-либо информации из модели.

В файле **/components/com_myquestions/controller.php** измените код методов класса **QuestionController** так:

```
function display()
{
  global $app;
  $pathway =& $app->getPathway();
  $pathway->addItem(JText::_('COM_MYQUESTIONS'),
   JRoute::_('index.php?option=com_myquestions'));
…
}

function showForm()
{
  global $app;
  $pathway =& $app->getPathway();
  $pathway->addItem(JText::_('COM_MYQUESTIONS'),
```

```
  JRoute::_('index.php?option=com_myquestions'));
  $pathway->addItem(JText::_('COM_MYQUESTIONS_ADD_QUESTION'));
…
}
```

Изменим метод отображения категории так, чтобы в навигационной цепочке выводилось название категории. Добавьте в метод QuestionViewCategory::display() в файле **/components/com_myquestions/views/category/view.html.php** код:

```
global $app;
$pathway =& $app->getPathway();
$pathway->addItem($name_cat);
```

Метод QuestionViewQuestion::display() в файле **/components/com_myquestions/views/question/tmpl/default.php** измените так:

```
function display($tpl=null)
{
  if ($tpl !== 'form')
  {
    global $option, $app;
    …
    $pathway =& $app->getPathway();
    $pathway->addItem($question->name_cat, $this->link_cat);
    $pathway->addItem(JString::substr(strip_tags($question->question), 0,
15).'...');
  }
  parent::display($tpl);
}
```

В случае запроса шаблона default_form в данном методе не требуется никаких изменений, так как необходимые элементы навигационной цепочки уже были добавлены в контроллере. По этой же причине не нужно ничего изменять в методе display() представления all.

Добавьте в файл **/language/ru-RU/ru-RU.com_myquestions.ini** строку:

```
COM_MYQUESTIONS="Моя система &laquo;вопрос &ndash; ответ&raquo;"
```

Навигационная цепочка, которая теперь отображается во фронтенде компонента, показана на рис. 7.13.

Рис. 7.13. Навигационная цепочка на странице с выводом одного вопроса

Ключевые термины

JPagination	- класс для вывода элементов формы для разбивки на страницы списков элементов.
JPathway	- класс для управления навигационной цепочкой.

Краткие итоги

По сравнению с разработкой компонента написать модуль значительно легче, так как он, как правило, не использует собственных таблиц и не обрабатывает данные, введенные пользователем.

Joomla позволяет разбивать длинные списки на страницы, задавая длину списка по умолчанию. Для вывода списков элементов с разбивкой на страницы как в бэкенде, и так и во фронтенде используется класс JPagination. Его методы генерируют HTML-код таких элементов, как счетчик страниц, ссылки на предыдущую/следующую страницу и т.д.

Для управления навигационной цепочкой в Joomla существует объект JPathway. Его методы позволяют добавлять элементы в навигационную цепочку, изменять названия отдельных элементов или весь массив целиком.

Вопросы

1. Почему разработать модуль легче, чем компонент?
2. Каким образом Joomla позволяет разбивать длинные списки на страницы?
3. Какой объект используется для управления навигационной цепочкой?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. список вариантов в дополнительных материалах).

8. Лекция: Файл-манифест

Рассмотрена структура файла-манифеста. Приведен практический пример создания установочного пакета для модуля и для компонента.

Цель лекции: Изучить структуру файла-манифеста и процесс создания установочных пакетов для модулей и компонентов.

Структура манифеста

Для каждого расширения Joomla может существовать файл-манифест. **Манифест** - это файл XML, содержащий метаданные о расширении, данные для установки и/или описание его настроек. Манифест должен называться <имя расширения>.xml и находиться в корневой директории установочного пакета.

Иерархия элементов в манифесте приведена на рис. 8.1.

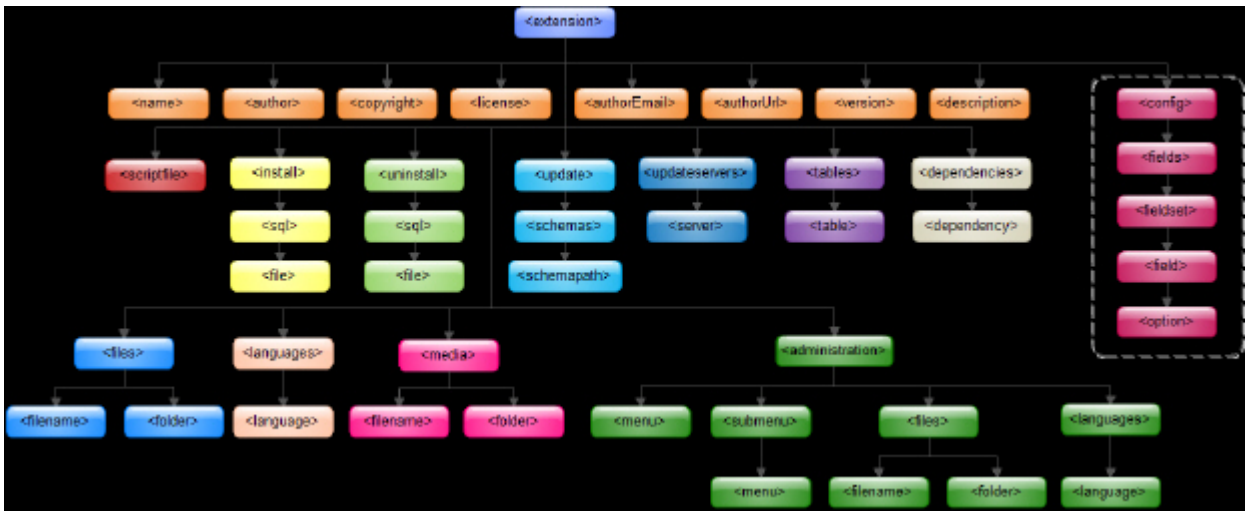


Рис. 8.1. Иерархия элементов в файле-манифесте

Корневым тегом манифеста является тег <extension></extension>. Он может иметь следующие атрибуты:

type	- тип расширения: component, file, language, library, module, package, plugin;
version	- версия Joomla, для которой написано расширение: 1.6, 2.5 и т.д.;
method	- будут ли при установке перезаписаны файлы расширения, если они уже существуют: upgrade (да), new (сообщить в таком случае об ошибке);
client	- для модулей: задает, предназначен этот модуль для бэкенда (administrator) или фронтенда (site);
group	- для плагинов: группа.

Внутри тега <extension> в первую очередь обычно задаются метаданные: <name>, <author>, <copyright>, <license>, <authorEmail>, <authorUrl>, <version>, <description>, однако их можно опустить.

Элемент <scriptfile> задает PHP-скрипт, который будет выполнен до, во время и/или после установки, удаления или обновления расширения. Этот скрипт должен содержать класс <префикс><имя расширения>IntallerScript, где префикс зависит от типа расширения (com_, mod_ и т.д.). Данный класс должен содержать следующие public-методы:

```
__constructor(JAdapterInstance $adapter);
bool preflight(string $route, JAdapterInstance $adapter);
bool postflight(string $route, JAdapterInstance $adapter);
bool install(JAdapterInstance $adapter);
bool update(JAdapterInstance $adapter);
bool uninstall(JAdapterInstance $adapter);
```

где

adapter	- объект, отвечающий за запуск этого скрипта;
$route	- какое событие происходит: install, uninstall, discover_install. Событие discover_install происходит при поиске расширений из менеджера расширений в панели управления

(эта функция Joomla позволяет установить расширения, файлы которых предварительно были загружены на сайт вручную).

Методы preflight() и postflight() будут вызваны соответственно до и после установки/удаления/обновления расширения.

Элементы <install> и <uninstall> задают SQL-скрипты, которые должны быть выполнены при установке и удалении расширения. Таким путем в базе данных создаются таблицы, хранящие данные, использующиеся расширением. Атрибут folder задает папку, в которой находятся эти скрипты в установочном пакете, например, "admin". В эти элементы должен быть вложен элемент <sql>, содержащий по одному элементу <file> для каждого файла SQL. Атрибуты тега <file>: driver - драйвер базы данных и charset - кодировка базы данных. Например:

```
<install>
  <sql>
    <file driver="mysql" charset="utf8">sql/install.sql</file>
  </sql>
</install>
<uninstall>
  <sql>
    <file driver="mysql" charset="utf8" folder="sql">sql/uninstall.sql</file>
  </sql>
</uninstall>
```

Элемент <files> задает список файлов, которые должны быть скопированы при установке расширения в соответствующую директорию во фронтенде. Для каждого файла добавляется вложенный элемент <filename>, для каждой папки - <folder>, причем содержимое папки уже не описывается. Пример для типичного компонента MVC:

```
<files folder="site">   <folder>models</folder>
<folder>views</folder>
  <folder>controllers</folder>
  <filename>mycomponent.php</filename>
<filename>router.php</filename>
</files>
```

Языковые файлы описываются внутри элемента <languages>. В установочном пакете эти файлы должны находиться в папке **/language/<код языка>**. Для каждого из них создается вложенный элемент <language> с атрибутом tag, содержащим код языка в формате <ln-LN>:

```
<languages folder="site">
         <language tag="ru-RU">language/ru-RU/ru-
RU.com_mycomponent.ini</language>
</languages>
```

Медиа-файлы - изображения, файлы Javascript и CSS, флэш - описываются внутри элемента <media>. Атрибут destination задает название папки, в которую будут скопированы перечисленные файлы. Эта папка должна находиться внутри папки **/media** в корне сайта. Если она не существует, то будет создана при установке расширения.

```
<media destination="com_mycomponent">
  <folder>css</folder>
  <folder>images</folder>
  <folder>js</folder>
```

```
  <filename>pic1.jpg</filename>
</media>
```

Файлы-манифесты компонентов могут включать элемент <administration>. Он содержит такие же элементы <languages> и <files>, как и описанные ранее, а также элементы для описания меню компонента в панели управления: <menu> и <submenu>.

Элемент <menu>, вложенный непосредственно в <administration>, описывает пункт в меню "**Компоненты**", ссылающийся на главную страницу бэкенда компонента. Атрибутом может быть img - относительный путь к пиктограмме пункта меню (по умолчанию будет присвоено значение "class:component").

Элемент <submenu> может содержать несколько вложенных элементов <menu>, описывающих подпункты этого пункта меню. У таких вложенных тегов <menu> может быть несколько атрибутов, среди которых в первую очередь отметим img и link - ссылка, переход по которой произойдет при щелчке на данном пункте меню. Если ссылка задана, то итоговый URL будет сформирован как index.php?<ссылка>. Если же атрибут link не задан, то установщик проверит наличие следующих атрибутов этого же тега: act, task, controller, view, layout, sub и добавит к строке "index.php?option=com_<имя компонента>&" соответствующие пары "имя-значение", разделенные амперсандом. Например, следующие элементы описывают пункт меню с одной и той же ссылкой **index.php?option=com_mycomponent&task=edit**:

```
<menu task="edit">COM_MYCOMPONENT_EDIT</menu>
<menu link="option=com_mycomponent&task=edit">COM_MYCOMPONENT_EDIT</menu>
```

Текст пункта меню задается внутри тега <menu> и обязательно должен быть ключом, значение которого должно быть переведено в языковом файле, обычно - /administrator/languages/<код языка>/<код языка>.com_<имя компонента>.sys.ini.

Элемент <config> описывает настройки расширения. Обратите внимание, что для компонентов этот элемент должен находиться не в файле-манифесте, а в отдельном файле **config.xml** в корневой директории бэкенда компонента. Для модулей <config> располагается в файле-манифесте и вложен в тег <extension>.

В <config> может быть вложен элемент <fields>. Его атрибут addfieldpath позволяет задать папку, в которой находится файл, определяющий собственный тип поля.

В <fields> или непосредственно в <config> вложен элемент <fieldset>, соответствующий HTML-элементу <fieldset>. Каждый <fieldset> - это группа элементов формы редактирования свойств расширения, например: "**Основные настройки**", "**Расширенные настройки**" и т.д. Атрибут name задает имя этого элемента, label - отображаемый текст.

Вложенные в <fieldset> элементы <field> задают поля формы. Каждый <field> соответствует одной настройке расширения. Допустимые атрибуты:

name	- имя
type	- тип
default	- значение по умолчанию

label	- текст, который будет выводиться рядом с соответствующим элементом управления на форме
description	- описание, которое будет выводиться во всплывающей подсказке.

Некоторые стандартные типы полей:

calendar	- текстовое поле, рядом с которым выводится пиктограмма для вывода календаря;
category	- выпадающий список категорий;
editors	- выпадающий список доступных WYSIWYG-редакторов;
filelist	- выпадающий список файлов из заданной директории;
folderlist	- выпадающий список папок из заданной директории;
imagelist	- выпадающий список файлов изображений из заданной директории;
languages	- выпадающий список установленных языковых файлов фронтенда или бэкенда;
list	- выпадающий список каких-либо строк;
password	- поле для ввода пароля;
radio	- переключатель;
spacer	- разделитель;
sql	- выпадающий список элементов, полученных в результате выполнения заданного SQL-запроса;
text	- текстовое поле;
textarea	- многострочное текстовое поле;
timezones	- список часовых поясов.

Для списков и переключателей в тег <field> должны быть вложены несколько элементов <option>, задающих значения, из которых пользователь может выбрать.

Например, так описывается выпадающий список для выбора порядка сортировки - по возрастанию или по убыванию:

```
<field name="entries_order" type="list"
default="DESC"
label="COM_MYCOMPONENT_ENTRIES_ORDER"
description="COM_MYCOMPONENT_ENTRIES_ORDER_DESC">
  <option value="DESC">COM_MYCOMPONENT_ENTRIES_ORDER_DESC</option>
  <option value="ASC">COM_MYCOMPONENT_ENTRIES_ORDER_ASC</option>
</field>
```

Для таких элементов, как filelist, folderlist, imagelist, вложенные элементы <option> не нужны, т.к. список значений определяется исходя из атрибутов этих тегов. Например, выпадающий список для выбора одного из доступных WYSIWYG-редакторов описывается так:

```
<field name="editors" type="editors" label="COM_MYCOMPONENT_EDITOR" />
```

Для создания собственного типа поля необходимо создать файл **<имя типа>.php** и в нем описать класс, производный от одного из классов подпакета Form: JFormFieldList, JFormFieldMedia и др. (полный список классов см. в документации). В классе должен быть

перегружен метод родительского класса, отвечающий за вывод поля формы. Простой пример приведен в документации Joomla.

Практика

В таблицах #__extensions и #__modules, в которые мы добавляли записи о наших расширениях, есть поле params, позволяющий хранить значения параметров. Например, для модуля mod_myquestions в этом поле хранится значение наподобие {"random":"0","items":"3","maxlen":"100","author":"1","date":"1"}. Нетрудно заметить, что данная строка представляет собой совокупность пар "**ключ - значение**". Таким образом, заданные значения настроек расширений сохраняются в базе данных. Наша задача - предоставить администратору сайта интерфейс для изменения этих значений.

Кроме того, мы создадим установочные пакеты для модуля и для компонента, представляющие собой ZIP-архивы определенной структуры.

Манифест для компонента

Настройки компонента

Зададим два параметра компонента myquestions - адреса электронной почты администратора сайта и эксперта.

Создайте файл **/administrator/components/com_myquestions/config.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<config>
   <fieldset name="component" label="COM_MYQUESTIONS_FIELDSETCONFIG_LABEL"
      description="COM_MYQUESTIONS_FIELDSETCONFIG_LABEL_DESC">
   <field name="email_admin" type="text"
label="COM_MYQUESTIONS_FIELD_EMAILADMIN_LABEL"
      description="COM_MYQUESTIONS_FIELD_EMAILADMIN_LABEL_DESC"
default="admin@mysite.ru"/>
   <field name="email_expert" type="text"
label="COM_MYQUESTIONS_FIELD_EMAILEXPERT_LABEL"
      description="COM_MYQUESTIONS_FIELD_EMAILEXPERT_LABEL_DESC"
default="expert@mysite.ru"/>
  </fieldset>
</config>
```

Добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.ini** код:

```
COM_MYQUESTIONS_CONFIGURATION="Настройки системы &laquo;вопрос &ndash;
ответ&raquo;"
COM_MYQUESTIONS_FIELDSETCONFIG_LABEL="Настройки системы"
COM_MYQUESTIONS_FIELDSETCONFIG_LABEL_DESC="Настройки системы"
COM_MYQUESTIONS_FIELD_EMAILADMIN_LABEL="E-mail администратора"
COM_MYQUESTIONS_FIELD_EMAILADMIN_LABEL_DESC="На этот адрес
будут приходить уведомления о новых вопросах, и с этого адреса будут
отправляться
уведомления экспертам и пользователям"
COM_MYQUESTIONS_FIELD_EMAILEXPERT_LABEL="E-mail эксперта"
COM_MYQUESTIONS_FIELD_EMAILEXPERT_LABEL_DESC="На этот адрес будут
приходить уведомления о новых вопросах, отправленные модератором"
```

Для отображения в панели инструментов кнопки "**Настройки**" используется метод JToolBarHelper::preferences(). Измените функцию TOOLBAR_myquestions::_DEFAULT() в файле **/administrator/components/com_myquestions/toolbar.myquestions.html.php** следующим образом:

```
function _DEFAULT()
  {
    JToolBarHelper::title(JText::_('COM_MYQUESTIONS_TOOLBAR_TITLE'),
'generic.png');
    JToolBarHelper::editList('reply','COM_MYQUESTIONS_REPLY');

JToolBarHelper::deleteList(JText::_('COM_MYQUESTIONS_TOOLBAR_REMOVE_QUESTIONS_CO
NFIRMATION'));
    JToolBarHelper::preferences('com_myquestions');
  }
```

Теперь на панели инструментов над списком вопросов появилась кнопка "**Настройки**", при нажатии на которую выводится окно, в котором можно задавать значения настроек компонента (рис. 8.2).

Рис. 8.2. Кнопка "Настройки" в панели инструментов

Внесем изменения в код компонента, чтобы использовать значения настроек. Измените код функции QuestionController::send() в файле **/administrator/components/com_myquestions/controller.php** следующим образом:

```
$mailer =& JFactory::getMailer();
  $params = JComponentHelper::getParams($option);
  $mailer->setSender($params->get('email_admin','admin@mysite.ru'));
  if ($task == 'sendToExpert')
  {
    $mailer->addRecipient($params->get('email_expert','expert@mysite.ru'));
    $mailer->setSubject(JText::_('COM_MYQUESTIONS_NEW_QUESTION'));
    $mailer->setBody(JText::sprintf('COM_MYQUESTIONS_EMAIL_EXPERT_BODY',$q));
  }
```

С помощью метода JComponentHelper::getParams() мы получаем объект JParameter, а затем используем его метод get() для получения параметров, задавая значения по умолчанию.

Изменим также код фронтенда. Откройте файл **/components/com_myquestions/controller.php** и измените функцию QuestionController::addQuestion() следующим образом:

```
…
    $mailer =& JFactory::getMailer();
    $option = JRequest::getVar('option','com_myquestions');
    $params = JComponentHelper::getParams($option);
    $mailer->setSender($params->get('email_admin','admin@mysite.ru'));
    $mailer->addRecipient($params->get('email_admin','admin@mysite.ru'));
…
```

Задайте с помощью кнопки "**Настройки**" собственные значения адресов электронной почты администратора сайта и эксперта. Добавьте на сайт вопрос, отправьте уведомление о нем эксперту, об ответе - автору вопроса и убедитесь, что в папке **<путь к Денверу>/tmp/!sendmail** появилось три письма с заданными вами значениями адресов электронной почты в соответствующих полях.

Упаковка компонента

Создайте в любом месте на диске новую папку, а в ней - папки **admin** и **site**, а также файл **myquestions.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<extension type="component" version="1.7">
   <name>com_myquestions</name>
   <author>Me</author>
   <creationDate>2012</creationDate>
   <copyright>Copyright (C) 2012 Me</copyright>
   <license>GNU General Public License version 2 or later</license>
   <authorEmail>admin@mysite.ru</authorEmail>
   <authorUrl>www.mysite.ru</authorUrl>
   <version>1.7.0</version>
   <description>COM_MYQUESTIONS_XML_DESCRIPTION</description>

    <install folder="admin">
        <sql>
            <file driver="mysql" charset="utf8">sql/install.sql</file>
        </sql>
    </install>
    <uninstall folder="admin">
      <sql>
        <file driver="mysql" charset="utf8">sql/uninstall.sql</file>
      </sql>
    </uninstall>
    <files folder="site">
      <filename>controller.php</filename>
      <filename>myquestions.html.php</filename>
      <filename>myquestions.php</filename>
      <filename>router.php</filename>
      <folder>models</folder>
      <folder>views</folder>
    </files>
    <languages folder="site">
      <language tag="ru-RU">language/ru-RU/ru-RU.com_myquestions.ini</language>
    </languages>
    <administration>
      <menu img="class:component">COM_MYQUESTIONS_MENU</menu>
```

```
    <submenu>
        <menu img="class:component"
link="option=com_myquestions">COM_MYQUESTIONS_MENU_QUESTIONS</menu>
        <menu img="class:component"
link="option=com_myquestions&amp;task=showcat">COM_MYQUESTIONS_MENU_CATEGORIES</
menu>
    </submenu>
      <files folder="admin">
        <filename>admin.myquestions.html.php</filename>
        <filename>admin.myquestions.php</filename>
        <filename>config.xml</filename>
        <filename>controller.php</filename>
        <filename>toolbar.myquestions.html.php</filename>
        <filename>toolbar.myquestions.php</filename>
        <folder>tables</folder>
        <folder>sql</folder>
      </files>
      <languages folder="admin">
        <language tag="ru-RU">language/ru-RU/ru-
RU.com_myquestions.ini</language>
        <language tag="ru-RU">language/ru-RU/ru-
RU.com_myquestions.sys.ini</language>
      </languages>
    </administration>
</extension>
```
Листинг .

Теги, задающие метаданные, такие как author, creationDate и другие, скопированы из файла **/modules/mod_myquestions/mod_myquestions.xml**.

Тег <install> задает SQL-файл, который будет выполнен при установке компонента. Обратите внимание, что явно указана его кодировка - utf-8 и, следовательно, этот файл нужно будет сохранить в указанной кодировке. Заметьте также, что тег <install> имеет атрибут folder со значением admin, что означает, что инсталлятор будет искать SQL-файл в папке **admin** установочного пакета. Кроме того, файл задан как sql/install.sql, то есть он находится в папке **sql** внутри папки **admin**. Тег <uninstall> имеет аналогичное содержимое.

Далее следует тег files со значением атрибута folder, равным site, т.е. описаны папки и файлы, которые находятся в папке **site** установочного пакета. Обратите внимание, что содержимое подпапок (**models** и **views**) не описывается, указывается только имя подпапки.

Содержимое тега <languages> описывает языковые файлы. Так как задан атрибут site, то эти файлы будут скопированы из папки **site** установочного пакета.

Оставшуюся часть файла **myquestions.xml** занимает тег <administration>. Вложенные в него теги <files> и <languages> аналогичны тегам, описывавшим фронтенд, за исключением атрибутов folder, имеющих значение admin. Тег <menu> описывает пункт меню, который появится в меню "**Компоненты**" панели управления, а также два его подпункта. Обратите внимание, что при задании ссылок для пунктов меню вместо амперсанда необходимо указывать его HTML-эквивалент &.

Добавьте в файл **/administrator/language/ru-RU/ru-RU.com_myquestions.sys.ini** строки:

```
COM_MYQUESTIONS="Моя система &laquo;вопрос &ndash; ответ&raquo;"
COM_MYQUESTIONS_XML_DESCRIPTION="Моя система &laquo;вопрос &ndash; ответ&raquo;"
```

В папку **admin** скопируйте содержимое папки **/administrator/components/com_myquestions**, а в папку **site - /components/com_myquestions**.

Кроме того, создайте в папке **site** папку **language**, в ней - папку **ru-RU**. Скопируйте в нее файл **ru-RU.com_myquestions.ini** из **language/ru-RU**.

Создайте в папке **admin** папку **language**, в ней - папку **ru-RU** и скопируйте в нее файлы **ru-RU.com_myquestions.sys.ini** и **ru-RU.com_myquestions.ini** из **/administrator/language/ru-RU**.

Наконец, создайте в папке **admin** подпапку **sql**, а в ней создайте файлы **install.sql** и **uninstall.sql**.

Получившееся дерево папок показано на рис. 8.3.

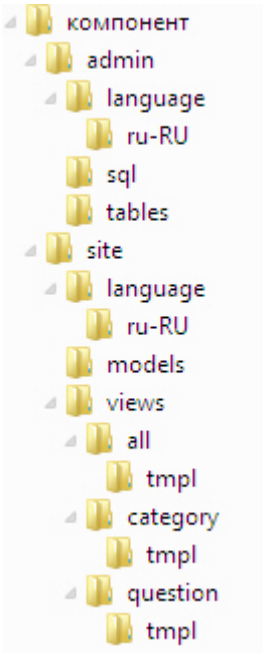


Рис. 8.3. Дерево папок установочного пакета

В файл **install.sql** вставьте код

```
DROP TABLE IF EXISTS `#__myquestions`; DROP TABLE IF EXISTS
`#__myquestions_categories`;
CREATE TABLE `#__myquestions`
(
`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
`name` VARCHAR(255) NOT NULL,
`date` DATETIME NOT NULL,
`question` TEXT NOT NULL,
`city` VARCHAR(50) NULL,
`email` VARCHAR(50) NOT NULL,
`IP` VARCHAR(15) NOT NULL,
`id_cat` INT NOT NULL,
`published` TINYINT(1) NULL DEFAULT '1',
`expiration_date` DATETIME NULL DEFAULT '0000-00-00 00:00:00',
`senttoexpert` TINYINT(1) NULL DEFAULT '0',
`answer` TEXT NULL DEFAULT '',
`senttoauthor` TINYINT(1) NULL DEFAULT '0'
);

CREATE TABLE `#__myquestions_categories`
(
`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
`name` VARCHAR(255) NOT NULL,
```

```
`desc` TEXT NOT NULL DEFAULT ''
);

INSERT INTO `#__myquestions_categories`(`name`, `desc`) VALUES('Без
категории','');
```

В сущности, это тот самый код, который использовался нами ранее для создания таблиц, но вместо реального префикса таблиц указан символический. До создания таблиц мы выполняем запросы для удаления таблиц с теми же названиями, если они существуют, чтобы избежать ошибок при переустановке компонента.

В таблицу #__myquestions_categories сразу же вставляется запись для категории под названием "**Без категории**". Так как таблица только что создана, эта запись получит id, равный 1, то совпадающий с id категории, который в нашем компоненте присваивается новому вопросу при добавлении.

Запросы из скрипта **uninstall.sql** удаляют обе таблицы:

```
DROP TABLE `#__myquestions`; DROP TABLE `#__myquestions_categories`;
```

Упакуйте папки **admin**, **site** и файл **myquestions.xml** в архив **com_myquestions.zip**. Установочный пакет компонента готов. Для проверки его работоспособности создайте новую установку Joomla и установите на нее ваш компонент с помощью менеджера расширений в панели управления.

Манифест для модуля

Настройки модуля

Создайте файл **/modules/mod_myquestions/mod_myquestions.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<extension type="module" version="1.7">
  <name>mod_myquestions</name>
  <author>Me</author>
  <creationDate>2012</creationDate>
  <copyright>Copyright (C) 2012 Me</copyright>
  <license>GNU General Public License version 2 or later</license>
  <authorEmail>admin@mysite.ru</authorEmail>
  <authorUrl>www.mysite.ru</authorUrl>
  <version>1.7.0</version>
  <description>MOD_MYQUESTIONS_XML_DESCRIPTION</description>
  <config>
    <fields name="params">
      <fieldset name="basic">
        <field name="random" type="radio" default="0"
label="MOD_MYQUESTIONS_RANDOMIZE_LABEL"
description="MOD_MYQUESTIONS_RANDOMIZE_LABEL_DESC">
          <option value="0">JNO</option>
          <option value="1">JYES</option>
        </field>
        <field name="items" type="text" default="1"
label="MOD_MYQUESTIONS_ITEMS_LABEL"
description="MOD_MYQUESTIONS_ITEMS_LABEL_DESC"/>
        <field name="maxlen" type="text" default="100"
```

```
label="MOD_MYQUESTIONS_MAXLEN_LABEL"
description="MOD_MYQUESTIONS_MAXLEN_LABEL_DESC"/>
        <field name="author" type="radio" default="1"
label="MOD_MYQUESTIONS_AUTHOR_LABEL"
description="MOD_MYQUESTIONS_AUTHOR_LABEL_DESC">
          <option value="0">JNO</option>
          <option value="1">JYES</option>
        </field>
        <field name="date" type="radio" default="1"
label="MOD_MYQUESTIONS_DATE_LABEL"
description="MOD_MYQUESTIONS_DATE_LABEL_DESC">
          <option value="0">JNO</option>
          <option value="1">JYES</option>
        </field>
      </fieldset>
    </fields>
  </config>
</extension>
```

Значения атрибутов тега <extension> определяют, что расширение, для которого написан этот манифест, - это модуль и что он написан для Joomla 1.7.

Описаны пять параметров модуля, три из которых - random, author и date - являются переключателями, а остальные два - items и maxlen - текстовыми полями. Для переключателей с помощью тега <option> задаются все возможные значения.

Теперь создайте языковой файл для модуля **/language/ru-RU/ru-RU.mod_myquestions.ini**:

```
MOD_MYQUESTIONS_XML_DESCRIPTION="Выводит вопросы из системы Вопрос - ответ"
MOD_MYQUESTIONS_RANDOMIZE_LABEL="Случайный вопрос"
MOD_MYQUESTIONS_RANDOMIZE_LABEL_DESC="Выводить один случайный вопрос?"
MOD_MYQUESTIONS_ITEMS_LABEL="Количество вопросов";
MOD_MYQUESTIONS_ITEMS_LABEL_DESC="Сколько вопросов выводить?"
MOD_MYQUESTIONS_MAXLEN_LABEL="Длина вопроса"
MOD_MYQUESTIONS_MAXLEN_LABEL_DESC="Сколько первых символов вопроса отображать?"
MOD_MYQUESTIONS_AUTHOR_LABEL="Автор вопроса"
MOD_MYQUESTIONS_AUTHOR_LABEL_DESC="Выводить имя автора?"
MOD_MYQUESTIONS_DATE_LABEL="Дата вопроса"
MOD_MYQUESTIONS_DATE_LABEL_DESC="Выводить дату написания вопроса?"
```

В панели управления перейдите в "**Расширения**" - "**Менеджер модулей**". Выберите из списка модуль "**Новые вопросы**", и в правой части открывшейся страницы будет отображена группа элементов формы для установки параметров модуля (рис. 8.4).

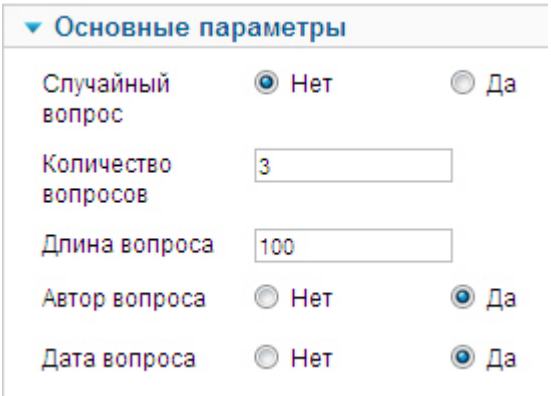


Рис. 8.4. Установка параметров модуля

Упаковка модуля

Для упаковки модуля добавьте в файл **/modules/mod_myquestions/mod_myquestions.xml** выделенный код:

```
<description>MOD_MYQUESTIONS_XML_DESCRIPTION</description>
  <files>
        <filename module="mod_myquestions">mod_myquestions.php</filename>
        <filename>mod_myquestions.xml</filename>
    </files>
    <languages>
        <language tag="ru-RU">language/ru-RU/ru-
RU.mod_myquestions.ini</language>
        <language tag="ru-RU">language/ru-RU/ru-
RU.mod_myquestions.sys.ini</language>
    </languages>
```

Создайте в папке **/modules/mod_myquestions** папку **language**, в ней - папку **ru-RU**. Скопируйте в нее файл **ru-RU.mod_myquestions.ini** из папки **language/ru-RU**. Создайте файл **/modules/mod_myquestions/language/ru-RU/ru-RU.mod_myquestions.sys.ini**:

```
MOD_MYQUESTIONS="Новые вопросы"
MOD_MYQUESTIONS_XML_DESCRIPTION="Выводит вопросы из системы Вопрос -
ответ"
```

Создайте из содержимого папки **/modules/mod_myquestions** архив **mod_myquestions.zip**. Это и есть установочный пакет для модуля. Обратите внимание, что в архиве должно находиться именно содержимое папки **mod_myquestions**, а не сама эта папка.

Ключевые термины

Манифест - файл XML, содержащий метаданные о расширении, данные для установки и/или описание его настроек.

Краткие итоги

Для каждого расширения Joomla может существовать файл-манифест, содержащий метаданные о расширении, данные для установки и/или описание его настроек. Манифест должен называться **<имя расширения>.xml** и находиться в корневой директории установочного пакета.

Настройки расширения описаны для модулей непосредственно в манифесте, а для компонентов - в отдельном файле **config.xml** в корневой директории бэкенда компонента.

Установочные пакеты для модуля и для компонента представляют собой ZIP-архивы определенной структуры.

Для компонента установочный пакет включает манифест и папки **admin** и **site**, содержащие файлы и папки бэкенда и фронтенда соответственно и, возможно, некоторые дополнительные файлы и папки, например, SQL-скрипты.

Для модуля установочный пакет просто включает все его папки и файлы.

Вопросы

1. Какие данные содержит манифест расширения Joomla?
2. Где описаны настройки модулей и компонентов?
3. Какова структура установочного пакет для модуля и для компонента?

Упражнения

Адаптируйте код из раздела "**Практика**" для своего варианта (см. список вариантов в дополнительных материалах).

Заключение

В рамках курса были рассмотрены основы программирования под CMS Joomla. Были предложены для изучения такие темы, как архитектура Joomla, работа с базой данных, генерация элементов HTML, организация иерархии пунктов меню, создание и отправка электронных писем, генерация SEF-ссылок, основы реализации архитектуры MVC с помощью Joomla, разработка простых модулей, организация постраничного вывода списков, управление навигационной цепочкой, создание манифестов расширений и установочных пакетов. Изучен ряд классов фреймворка Joomla. Таким образом, успешное освоение материалов данного курса достаточно для разработки небольших компонентов и модулей для Joomla.

Общий глоссарий

JAdministrator	- приложение, управляющее функциями для администрирования Joomla.
JApplication	- класс, позволяющий работать с очередью сообщений, осуществлять перенаправление браузера, получать параметры конфигурации сайта, определять тип запущенного приложения Joomla.
JController	- абстрактный класс для реализации контроллеров.
JDatabase	- абстрактный класс, предоставляющий доступ к соединению с базой данных, создающемуся при инициализации приложения Joomla.
JDatabaseQuery	- класс, методы которого совпадают с ключевыми словами языка SQL и позволяют упростить создание сложных SQL-запросов.
JDate	- класс для работы с датами.
JDocument	- класс для работы с документом.
JEditor	- класс для работы с WYSIWYG-редактором.
JError	- класс для работы с ошибками.
JFactory	- класс Joomla, реализующий паттерн "фабрика" и позволяющий получить доступ к глобальным объектам фреймворка.
JHTML	- класс для вывода элементов XHTML.
JHTMLBehavior	- поддерживающий класс, который позволяет вывести календарь, дерево элементов, файловый загрузчик и некоторые другие элементы управления.

JHTMLEmail	- поддерживающий класс, содержащий метод для скрытия адреса электронной почты в целях его защиты от спам-ботов.
JHTMLForm	- поддерживающий класс, содержащий метод, который возвращает код скрытого поля формы для уменьшения риска CSRF-атак.
JHTMLGrid	- поддерживающий класс, позволяющий вывести в таблице в панели управления такие элементы, как чекбокс, пиктограмма для переключения состояния "опубликовано"/"не опубликовано", отобразить заголовок столбца как ссылки для сортировки по этому столбцу и др.
JHTMLImage	- поддерживающий класс, содержащий методы для поиска изображения в фронтенде и бэкенде.
JHTMLList	- поддерживающий класс для создания списков некоторых конкретных значений.
JHTMLSelect	- поддерживающий класс для генерации кода списков.
JInstallation	- приложение, которое запускается при установке Joomla.
JMail	- класс для создания и отправки электронных писем.
JMailHelper	- класс для очистки данных перед добавлением к электронному письму и проверки, является ли заданная строка корректным адресом электронной почты.
JModel	- абстрактный класс для реализации моделей.
JPagination	- класс для вывода элементов формы для разбивки на страницы списков элементов.
JPathway	- класс для управления навигационной цепочкой.
JRequest	- класс Joomla, использующийся для работы с переменными HTTP-запроса.
JRoute	- класс для создания SEF-ссылок.
JSite	- приложение, отвечающее за компоновку и отображение фронтенда.
JTable	- класс, реализующий паттерн Active Record и использующийся для управления таблицами базы данных.
JToolBarHelper	- класс Joomla, содержащий методы, которые генерируют HTML-код для построения кнопок панелей инструментов.
JURI	- класс для работы с URI.
JUser	- класс для работы с данными о пользователе.
JView	- абстрактный класс для реализации представлений.
XML-RPC	- приложение, позволяющее администрировать сайт Joomla удаленно.
Библиотека	- файл, который требуется для работы фреймворка или сторонних расширений.
Бэкенд	- система администрирования сайта.
Документ	- буфер, использующийся для хранения содержимого веб-страницы, которая будет показана пользователю после выполнения запроса.
Иерархия пунктов меню	- дерево, состоящее из пунктов меню и организованное с помощью вложенных множеств.
Ключ	- эквивалент текста, подлежащего переводу.

Компонент	- основной тип расширений Joomla, вызов которого происходит при каждом обращении к Joomla.
Манифест	- файл XML, содержащий метаданные о расширении, данные для установки и/или описание его настроек.
Модуль	- расширение Joomla, использующееся для отображения небольших фрагментов контента, обычно в левой или правой колонке или верхней или нижней областях страницы.
Основной метод класса JHTML	- метод JHTML::_(), который вызывает метод, определяющийся его первым параметром, и передает ему свои остальные параметры.
Очередь сообщений	- массив строк, которые будут выведены на экран при следующей загрузке какой-либо страницы.
Перевод	- строка, содержащая перевод текста, соответствующего заданному ключу, на какой-либо язык.
Плагин	- расширение Joomla, позволяющее зарегистрировать функции и классы для обработки каких-либо событий, вызванных Joomla, например, поиск по сайту.
Поддерживающие классы	- классы для вывода элементов XHTML и поведений Javascript.
Префикс таблиц базы данных	- строка, которая присоединяется к названию каждой таблицы Joomla в базе данных.
Приложение	- глобальный объект, использующийся для обработки запросов.
Реальный префикс	- то конкретное сочетание символов, которое используется в названиях таблиц базы данных.
Регистрация задачи	- сопоставление ее какому-либо методу класса, производного от JController.
Связывание	- процесс присвоения каждому полю производного от JTable класса значения элемента массива переменных запроса, так что ключ элемента совпадает с названием поля.
Символический префикс	- сочетание "#__" (решетка и два знака подчеркивания), которое используется в запросах вместо реального префикса.
Уровень приложения	- часть архитектуры Joomla, которая состоит из приложений, расширяющих абстрактный класс JApplication.
Уровень расширений	- часть архитектуры Joomla, которая состоит из расширений фреймворка Joomla и приложений.
Уровень фреймворка	- часть архитектуры Joomla, которая обеспечивает ее базовую функциональность с помощью набора библиотек и плагинов и собственно ядра Joomla.
Фреймворк Joomla ("ядро")	- набор классов, обеспечивающих базовую функциональность Joomla (JDatabase, JUser, JForm, JEditor и т.д.).
Фронтенд	- часть сайта, доступная пользователю.
Функция генерации SEF-ссылок	- функция, которая принимает массив элементов HTTP-запроса и возвращает массив сегментов SEF-ссылки.
Функция декодирования SEF-ссылок	- функция, которая из массива сегментов SEF-ссылки создает массив переменных HTTP-запроса.

Шаблон	- расширение Joomla, отвечающее за внешний вид сайта.
Шаблон SEF-ссылок	- последовательность сегментов.
Языковый файл	- расширение Joomla, позволяющее представить ее контент на нескольких языках.

Список сокращений

CCK	Content Construction Kit	Конструктор контента
CMS	Content Management System	Система управления контентом
MVC	Model - View - Controller	Архитектура "Модель - Представление - Контроллер"
SEF	Search Engine Friendly	Ссылка, удобная для восприятия поисковыми системами