





Joomla!

Программирование

Марк Декстер
Луис Лэндри



Joomla!

Программирование

Сканирование и обработка LUCKY RAT
Вопросы и пожелания olegvedehin@yandex.ru

Joomla!™ Programming

Mark Dexter
Louis Landry

Addison-Wesley

Joomla!
Программирование

Марк Декстер
Луис Лэндри



Москва • Санкт-Петербург • Киев
2013

ББК 32.973.26-018.2.75
Д28
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С.Н. Тригуб*
Перевод с английского и редакция *И.В. Берштейна*

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

Декстер, Марк, **Лэндри,** Луис.

Д28 Joomla!: программирование. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2013. — 592 с.: ил. — Парал. тит. англ.

ISBN 978-5-8459-1798-0 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Addison-Wesley Publishing Company, Inc.

Authorized translation from the English language edition published by Addison-Wesley Publishing Company, Inc, Copyright © 2012 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form, by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

Russian language edition is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2013.

Научно-популярное издание

Марк Декстер, Луис Лэндри
Joomla!: программирование

Литературный редактор *И.А. Попова*
Верстка *Л.В. Чернокозинская*
Художественный редактор *В.Г. Павлютин*
Корректор *Л.А. Гордиенко*

Подписано в печать 08.01.2013. Формат 70x100/16
Гарнитура Minion. Печать офсетная.
Усл. печ. л. 37,0. Уч.-изд. л. 34,7.
Тираж 1500 экз. Заказ № 3469

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9-я линия, 12/28

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

ISBN 978-5-8459-1798-0 (рус.)
ISBN 978-0-13-278081-0 (англ.)

© Издательский дом "Вильямс", 2013
© Pearson Education, Inc., 2012

Оглавление

Благодарности 19

Об авторах 21

Предисловие 23

1. Основы разработки в Joomla! 31

2. Подготовка рабочей станции к разработке в Joomla! 49

3. Принцип действия Joomla! 73

4. Расширение Joomla замещениями компоновки 125

5. Расширение Joomla подключаемыми модулями 173

6. Расширение Joomla модулями 225

7. Компоненты, часть I. Контроллеры и модели 267

8. Компоненты, часть II. Представления, класс `JForm` и пользовательская часть 305

9. Компоненты, часть III. Пример разработки административной части компонента 351

10. Компоненты, часть IV. Пример разработки пользовательской части компонента 399

11. Работа с базой данных 447

12. Применение JavaScript и MooTools в Joomla! 487

13. Применение платформы Joomla! в качестве среды для разработки приложений 521

Приложение А. Краткий курс PHP и ООП 545

Приложение Б. Типы фильтров в Joomla! 569

Приложение В. Методы из класса `JHtml` 573

Приложение Г. Словарь специальных терминов 579

Предметный указатель 585

Содержание

Благодарности 19

Об авторах 21

Предисловие 23

Опытные пользователи Joomla, только начинающие программировать 23

Опытные программисты, только начинающие осваивать Joomla 24

Разработчики, которым требуется быстро освоить версию 2.5 Joomla 24

Читатели, которым требуется узнать больше о Joomla и разработке расширений 24

Вопросы, не рассматриваемые в книге 24

Структура книги 25

Трудности веб-разработки — нужно слишком много знать! 25

Нововведения в версии Joomla 2.5 26

Система списков контроля доступа 27

Определяемые пользователем уровни категорий (и больше никаких разделов) 27

Класс `JForm` 27

Класс `JTableNested` 27

Класс `JDatabaseQuery` 27

Версия PHP 5.2 28

Версия MySQL 5.0.4 28

Формат файлов языковой поддержки 28

Упрощенная процедура обновления 28

Усовершенствования шаблона MVC 28

Поддержка других баз данных 29

Усовершенствованный поиск 29

Веб-сайт, сопровождающий книгу 29

Добро пожаловать в сообщество разработчиков Joomla 29

От издательства 30

1. Основы разработки в Joomla! 31

Разрабатывать в Joomla совсем не трудно! 31

Техническая среда Joomla 31

Требования к программированию для Joomla 33

Способы расширения Joomla 34

Открытость кода означает полный контроль над ним 34

Изменения на веб-странице с помощью файлов замещения 35

Расширения 36

Подключаемые модули 37

Модули 39

Компоненты 40
Языковые расширения 42
Шаблоны 44
Выбор типа расширения для конкретного проекта 46
Применение Joomla в качестве платформы или интегрированной среды 46
Резюме 47

2. Подготовка рабочей станции к разработке в Joomla! 49

Требования к запуску Joomla 49
Папка `DocumentRoot` в Apache 50
Получение своевременных инструкций 51
Платформа Windows 51
Платформа Mac OS X 52
Платформа Linux 52
Владелец файлов и папок по умолчанию 53
Рабочие инструменты 53
Интегрированная среда разработки в сравнении с текстовым редактором 53
Варианты IDE с открытым кодом 54
Коммерческие IDE 68
Текстовые редакторы 68
Другие инструментальные средства 69
Программное обеспечение контроля версий 69
Программное обеспечение автоматического построения проектов (Ant и Phing) 70
Программы автоматизации тестирования 70
Резюме 71

3. Принцип действия Joomla! 73

Краткий обзор папок Joomla 74
Пользовательская и административная части 74
Папка `cache` 75
Папка `cli` 75
Папка `components` 76
Папка `images` 77
Папка `includes` 78
Папка `installation` 79
Папка `language` 79
Папка `libraries` 80
Папка `logs` 80
Папка `media` 80
Папка `modules` 80
Папка `plugins` 82
Папка `templates` 83
Папка `tmp` 83

Папка `administrator` 83
Платформа Joomla 89
Веб-программирование в сравнении с обычным программированием 91
Сохранение состояния программы 92
Контроль и проверка команд 94
Внутреннее строение цикла выполнения команд в Joomla 96
Загрузка файла `index.php` 96
Проверка среды выполнения 97
Определение мест расположения файлов 98
Загрузка интегрированной среды Joomla 101
Начало или продолжение сеанса 102
Маршрутизация URL 103
Выполнение компонента 105
Воспроизведение страницы 111
Вывод страницы 116
Краткие итоги сеанса работы с веб-сайтом Joomla 117
Условные обозначения имен 119
Глобальные объекты 119
Краткий обзор таблиц базы данных 120
О префиксах имен таблиц 121
Резюме 124

4. Расширение Joomla замещениями компоновки 125

Основы применения шаблонов 125
Папки и файлы шаблонов 126
Файл шаблона `index.php` 127
Места расположения в шаблонах 130
Параметры настройки шаблонов 134
Стилевая рамка модуля 137
Копирование шаблона 139
Замещение компоновки модуля последних статей в шаблоне 142
Настройка модуля в выборочных данных 142
Создание файла замещения компоновки 144
Специальная настройка компоновки 145
Устранение ошибки с помощью функции `strip_tags()` 148
Применение метода `truncate()` из класса `JHtmlString` 150
Применения синтаксиса подчеркивания 154
Изменение внешнего вида компонента регистрации пользователя 155
Альтернативные компоновки 159
Добавление новой компоновки пункта меню 160
Переопределение параметров 161
О принципе действия замещений 163

Замещение других элементов оформления веб-сайта 164
Стилевая рамка модуля: добавление нового стиля оформления модуля 164
Языковые замещения: добавление перевода в замещение 169
Замещения таблиц и моделей 172
Резюме 172

5. Расширение Joomla подключаемыми модулями 173

Что такое подключаемый модуль 173
Принцип действия подключаемых модулей 174
Условные обозначения подключаемых модулей 174
Типы подключаемых модулей и место для их ввода 175
Аутентификация 175
Контрольное изображение 175
Содержимое 175
Редакторы 175
Editors-XTD 176
Расширение 176
Поиск 176
Интеллектуальный поиск 176
Система 176
Пользователь 176
Краткий обзор избранных базовых подключаемых модулей 177
Системный подключаемый модуль SEF 177
Подключаемый модуль аутентификации 180
Подключаемый модуль содержимого 189
Подключаемый модуль для обработки события `onBeforeCompileHead` 194
Подключаемый модуль для регистрации пользователей 200
Обновление файла замещения подтверждения 200
Ввод исходного XML-файла 201
Ввод исходного PHP-файла 202
Ввод файлов языковой поддержки 204
Проверка подключаемого модуля 206
Упаковка подключаемого модуля 207
Усовершенствованный подключаемый модуль для регистрации пользователей 209
Создание главного XML-файла для подключаемого модуля 210
Создание XML-файла для разметки формы 210
Создание исходного файла PHP для подключаемого модуля 212
Ввод файлов языковой поддержки 213
Проверка подключаемого модуля 214
Упаковка подключаемого модуля 215
Добавление параметров в подключаемый модуль 215
Применение подключаемых модулей для замещения основных классов 218
Каким образом импортируются подключаемые модули 218

Каким образом загружаются классы 219
Пример замещения класса `JTableNested` 220
Передовой опыт разработки и применения подключаемых модулей 222
Резюме 223

6. Расширение Joomla модулями 225

Что такое модуль 225
Модули в сравнении с компонентами 225
Краткий обзор базового модуля 226
XML-файл модуля 227
Главный файл модуля 227
Вспомогательный класс модуля 230
Файл исходной компоновки 234
Отображение статей текущего автора 236
Структура модуля 236
XML-файл модуля 237
Исходный файл PHP 241
Вспомогательный файл 243
Файл исходной компоновки 254
Файлы языковой поддержки 255
Проверка достоверности параметров средствами класса `JForm` 256
Справочный файл 264
Упаковка модуля 265
Краткий обзор созданного модуля 265
Передовой опыт разработки и применения модулей 266
Резюме 266

7. Компоненты, часть I. Контроллеры и модели 267

Что такое компонент 267
Основные функции компонентов 268
Своеобразие компонентов 268
Шаблон проектирования MVC 269
Административная часть компонента Weblinks 270
Установочные файлы 271
Меню компонентов 272
Параметры компонента 273
Вспомогательные классы и методы 273
Точка входа в компонент Weblinks 274
Функции контроллера в компоненте Weblinks 276
Модели компонента Weblinks 292
Табличный класс компонента Weblinks 300
Резюме 303

8. Компоненты, часть II. Представления, класс `JForm` и пользовательская часть 305

Представления и методы отображения 305
Представление списка веб-ссылок 306
Файл исходной компоновки 309
Представление одиночных веб-ссылок 317
Применение класса `JForm` в компоненте Weblinks 318
Сохранение объекта типа `JForm` в оперативной памяти 323
Видоизменение форм в динамическом режиме 324
Воспроизведение формы 325
Краткие итоги рассмотрения административной части компонента Weblinks 331
Пользовательская часть компонента Weblinks 331
Сходство структуры файлов и применения шаблона MVC 331
Типы пунктов меню 334
Маршрутизация в пользовательской части веб-сайта 337
Пользовательское представление ленты новостей 348
Резюме 349

9. Компоненты, часть III. Пример разработки административной части компонента 351

Краткий обзор назначения создаваемого примера компонента 351
Детальная разработка компонента 352
Файлы из административной части компонента 353
Экран Subscriptions Manager: Subscriptions 355
Исходный контроллер 356
Контроллер подчиненного диспетчера и задачи на панели инструментов 358
Представление диспетчера 360
Вспомогательный класс 364
Модель диспетчера 365
Таблицы базы данных 370
Компоновка экрана диспетчера 372
Экран Subscriptions Manager: Add and Edit 376
Задачи контроллера 377
Представление ввода и редактирования подписки 379
Модель ввода и редактирования подписок 381
Форма ввода и редактирования подписок 386
Табличный класс 391
Файлы языковой поддержки 394
Установка и настройка компонента 396
Резюме 398

8. Компоненты, часть II. Представления, класс `JForm` и пользовательская часть 305

Представления и методы отображения 305
Представление списка веб-ссылок 306
Файл исходной компоновки 309
Представление одиночных веб-ссылок 317
Применение класса `JForm` в компоненте Weblinks 318
Сохранение объекта типа `JForm` в оперативной памяти 323
Видоизменение форм в динамическом режиме 324
Воспроизведение формы 325
Краткие итоги рассмотрения административной части компонента Weblinks 331
Пользовательская часть компонента Weblinks 331
Сходство структуры файлов и применения шаблона MVC 331
Типы пунктов меню 334
Маршрутизация в пользовательской части веб-сайта 337
Пользовательское представление ленты новостей 348
Резюме 349

9. Компоненты, часть III. Пример разработки административной части компонента 351

Краткий обзор назначения создаваемого примера компонента 351
Детальная разработка компонента 352
Файлы из административной части компонента 353
Экран Subscriptions Manager: Subscriptions 355
Исходный контроллер 356
Контроллер подчиненного диспетчера и задачи на панели инструментов 358
Представление диспетчера 360
Вспомогательный класс 364
Модель диспетчера 365
Таблицы базы данных 370
Компоновка экрана диспетчера 372
Экран Subscriptions Manager: Add and Edit 376
Задачи контроллера 377
Представление ввода и редактирования подписки 379
Модель ввода и редактирования подписок 381
Форма ввода и редактирования подписок 386
Табличный класс 391
Файлы языковой поддержки 394
Установка и настройка компонента 396
Резюме 398

10. Компоненты, часть IV. Пример разработки пользовательской части компонента 399

Краткий обзор используемых файлов 399
Установочный XML-файл 400
Точка входа в компонент 401
Исходный контроллер 401
Представление категории подписок 403
XML-файл для разметки пункта меню 403
Представление категории 405
Модель 410
Вспомогательный файл категорий 414
Файлы компоновки категорий 415
Представление подписки 419
Методы контроллера для редактирования подписки 420
Представление и форма редактирования 422
Компоновка формы редактирования 425
Задача подписки 426
Модель для формы 429
Компоновка экрана с благодарностями за подписку 433
Файл языковой поддержки 435
Упаковка компонента 437
Новая функция: отчет о подписчиках в административной части компонента 437
Новая кнопка панели инструментов 438
Метод контроллера для выполнения новой задачи 439
Класс новой модели 439
Метод контроллера для создания файла экспорта 443
Составление отчета 444
Практические соображения 445
Резюме 446

11. Работа с базой данных 447

Краткий обзор баз данных 448
Создание и видоизменение таблицы по командам DDL 449
Команда `CREATE TABLE` 450
Типы данных 453
Атрибуты столбцов 455
Команда `ALTER TABLE` 456
Команда `DROP TABLE` 457
Применение программы phpMyAdmin 457
Тестирование и отладка запросов SQL 457
Создание сценариев DDL 459
Резервное и обычное копирование базы данных 461

Формирование запросов SQL к базе данных с помощью команд DML 462
Запросы по команде SELECT 462
Запросы по команде UPDATE 470
Запросы по команде INSERT 471
Запросы по команде DELETE 471
Запросы по команде UNION 472
Выражения в запросах 472
Разработка структуры таблиц 472
Справочные таблицы 473
Поля ключей и внешние ключи 473
Таблицы соответствий 474
Таблицы предыстории 475
Работа с базой данных в среде Joomla 475
Применение класса JDatabaseQuery 476
Обработка данных по запросу 482
Обработка запросов по командам INSERT, UPDATE и DELETE 482
Обработка запросов по команде SELECT 482
Резюме 485

12. Применение JavaScript и MooTools в Joomla! 487

Общее представление о JavaScript 487
Принцип действия JavaScript 488
Общее представление о MooTools 489
Порядок применения JavaScript и MooTools в Joomla 490
Встроенные средства JavaScript 491
Поведение calendar 492
Поведение caption 493
Поведение colorpicker 493
Проверка достоверности формы 494
Поведение framework 498
Поведение highlighter 498
Поведение keepalive 498
Поведение modal 499
Виды поведения checkall и multiselect 504
Поведение noframes 505
Поведение switcher 506
Поведение tooltip 509
Поведение tree 510
Поведение uploader 512
Применение расширений MooTools 513
Применение AJAX в Joomla 514
Применение других интегрированных сред JavaScript 518
Резюме 520

13. Применение платформы Joomla! в качестве среды для разработки приложений 521

Общее представление о платформе Joomla 521
Причины выделения платформы Joomla в отдельный проект 522
Назначение платфомы Joomla 522
Примеры разработки программ на платформе Joomla 523
Подготовка проекта на платформе 523
Приложение командной строки Hello World 525
Веб-приложение Hello WWW 526
Пример текущего контроля подписок 528
Структура проекта 529
Конфигурационный файл 529
Контрольный файл 530
Файл текущего контроля подписок 532
Выполнение приложения для текущего контроля подписок 542
Выполнение программ командной строки в системе Joomla CMS 543
Резюме 544

Приложение А. Краткий курс PHP и ООП 545

Структура файлов PHP 545
Файлы, содержащие только исходный код PHP 545
Файлы, содержащие исходный код PHP и HTML 545
Основы синтаксиса языка PHP 546
Наиболее важные символы 546
Наиболее употребительные операторы 548
Арифметические и строковые операторы 548
Задание и приведение типов переменных 548
Логические операторы 549
Условные операторы 549
Оператор выбора `switch` 551
Операторы цикла 551
Циклы `foreach` 551
Циклы `for` 552
Циклы `do/while` 552
Оператор `continue` 553
Альтернативный синтаксис 553
Переменные 554
Объявление переменных 554
Область действия переменных 554
Массивы 555
Обращение с массивами 556
Символьные строки 556

Константы и текущий каталог 556
Функции и методы 557
Структура функции 557
Область действия переменной в функции 558
Передача значений по ссылке 558
Включение файлов и их типы 559
Файлы объявления классов 559
Файлы объявления функций 559
Простые сценарии 560
Смешанные файлы 560
Включение файлов 560
Основы ООП 561
Классы и объекты 561
Метод-конструктор 561
Создание объектов 561
Стандартный класс 562
Расширение классов и наследование 562
Переопределение методов 563
Модификаторы `public`, `protected` и `private` 563
Статические методы, поля и переменные 563
Переменная `$this` и ключевые слова `self` и `parent` 564
Простая отладка программ 565
Просмотр определенных переменных 565
Просмотр трассировки стека 565
Более совершенные приемы программирования 566
Применение логического оператора `||` вместо условного оператора `if` 566
Связывание методов в цепочку 566
Волшебные методы PHP 567
Переменные в качестве имен классов и методов 568
Регулярные выражения 568

Приложение Б. Типы фильтров в Joomla! 569

Фильтрация элементов разметки в коде HTML 570
Фильтрация данных в приложениях Joomla 570
Фильтрация в классе `JForm` 570
Фильтрация в классах `JRequest` и `JInput` 571
Непосредственное применение класса `JFilterInput` для фильтрации данных 572

Приложение В. Методы из класса `JHtml` 573

Вызов методов из класса `JHtml` 573
Специальные классы типа `JHtml` 574
Методы из класса `JHtml` 574
Метод `link()` 574

Метод image() 574
Метод stylesheet() 574
Метод script() 575
Метод calendar() 575
Метод date() 575
Классы из папки html 575
Класс JHtmlBatch 575
Класс JHtmlBehavior 576
Класс JHtmlCategory 576
Класс JHtmlContent 576
Класс JHtmlContentLanguage 576
Класс JHtmlEmail 576
Класс JHtmlForm 576
Класс JHtmlGrid 577
Класс JHtmlImage 577
Класс JHtmlJGrid 577
Класс JHtmlList 577
Класс JHtmSelect 577
Класс JHtmSliders 578
Класс JHtmString 578
Класс JHtmTabs 578

Приложение Г. Словарь специальных терминов 579

Предметный указатель 585

Эта книга посвящается
многим трудолюбивым добровольцам
из сообщества разработчиков Joomla!,
чья искренняя преданность делу
постоянно питает веру авторов в человечество.

Благодарности

Сообщество разработчиков Joomla! представляет собой собрание дружелюбных, опытных специалистов, всегда готовых прийти на помощь начинающим. Эта книга не состоялась бы без добровольной помощи многих членов данного сообщества, любезно отвечавших на мои вопросы и помогавших мне восполнить пробел в знании предмета книги. Особую благодарность хотелось бы выразить тем, кто читал рукопись книги и дал неоценимые советы в откликах на нее, в том числе Андреа Тарру (Andrea Tarr), Элину Уорингу (Elin Waring), Омару Рамосу (Omar Ramos) и Сэму Моффатту (Sam Moffatt). И наконец, благодарю свою жену, Деб, за моральную поддержку и ободрение (а иногда и просто всякую женскую чепуху) в работе над книгой.

Марк Декстер

Об авторах

Марк Декстер занимается программированием с 1970-х годов. Он основал коммерческую компанию по разработке программного обеспечения и руководил ею в течение двадцати восьми лет, прежде чем выйти в отставку и посвятить себя добровольной работе над открытым программным обеспечением. Марк начал пользоваться Joomla! с 2008 года, присоединившись к Ведущей производственной группе (Production Leadership Team) в 2009 году. Он немало потрудился на самых разных участках проекта Joomla!, включая форумы пользователей и разработчиков, инициативную программу Google Summer of Code компании Google, составление документации и экранов оперативной справки, а также игру Bug Squad (Санитарный отряд). Марк принимает активное участие в процессе разработки Joomla с 2009 года, занимаясь исправлением программных ошибок, внедрением дополнительных средств и согласованием выпусков новых версий. Марк проживает в г. Сиэтл, шт. Вашингтон.

Луис Лэндри написал большую часть интегрированной среды Joomla для версий 1.5 и 1.6 и в течение пяти лет был одним из главных разработчиков архитектуры Joomla. Свой первый опыт общения с компьютером он начал с программирования на языке Basic, пользуясь ПК на процессоре Intel 286 в конторе своего отца. С тех пор Луис приобрел немалый опыт программирования на многих языках, включая низкоуровневые языки типа ассемблера x86, высокоуровневые вроде Java и языки написания сценариев, подобные PHP. Он также основал робототехническую группу в своем университете и стал ее активным участником. Луис — заядлый автомобилист и любит возиться с автомашинами в свободное от работы время. Он проживает в Кремниевой Долине, шт. Калифорния.

Предисловие

Разработка в Joomla! охватывает довольно широкий круг задач. Один проект может быть связан с созданием единственного файла замещения с целью изменить способ представления веб-страницы, а другой проект — с созданием расширения со многими модулями, компонентами и подключаемыми модулями.

И хотя в одной книге невозможно охватить все вопросы, интересующие разработчиков веб-приложений, тем не менее, немало полезного почерпнут для себя самые разные категории читателей: от начинающих пользоваться Joomla до опытных разработчиков, которым требуется быстро освоить версию 2.5.

Опытные пользователи Joomla, только начинающие программировать

К этой категории относятся читатели, которые, вероятно, уже сталкивались с ситуациями, когда одной только настройки параметров недостаточно для того, чтобы веб-сайт действовал именно так, как нужно. А может быть, им требуется расширение, не похожее на те, что имеются в каталоге расширений Joomla Extensions Directory (JED). В таком случае эта книга поможет приступить к специальной настройке Joomla с написания кода на языке PHP. И для этого совсем *не* требуется опыт программирования. Ведь основное назначение Joomla — дать возможность создавать веб-сайты, не имея ни знаний, ни опыта программирования на PHP или работы с базой данных MySQL, а также выполнять большую часть специальной настройки, опираясь на весьма скромные знания и опыт разработки веб-сайтов. Вы будете приятно удивлены теми результатами, которых позволяет добиться написание даже небольшого объема кода, а также тем, насколько быстро вы научитесь выяснять, что именно требуется для расширения гибкости и функциональности Joomla.

В этой книге предполагается, что у вас вообще нет никакого опыта программирования на PHP или работы с базой данных MySQL, и поэтому рассматриваемые в ней темы поясняются с самых основ. Кроме того, делаются ссылки на свободно доступные ресурсы, помогающие изучить упоминаемые темы более углубленно.

Опытные программисты, только начинающие осваивать Joomla

К этой категории относятся читатели, имеющие довольно основательную техническую подготовку, но стремящиеся разобраться в том, как действует Joomla. Эта книга организована таким образом, что вы сможете очень быстро найти в ней нужную информацию по данному вопросу. И хотя в книге даются основы программирования на PHP и работы с базой данных MySQL, вы можете без особого ущерба пропустить то, что уже знаете, чтобы сосредоточиться на конкретных сведениях о Joomla. Кроме того, в книге рассматриваются различные варианты выбора проектных решений, исходя из общей архитектуры Joomla, чтобы стало понятнее, почему конкретная программа была построена именно так, а не иначе.

Разработчики, которым требуется быстро освоить версию 2.5 Joomla

Версия Joomla 1.6 претерпела существенные изменения по сравнению с версией 1.5, особенно в отношении разработки. А в версии 1.7 и 2.5 были внесены очередные эволюционные изменения по сравнению с версией 1.6. Материал этой книги опирается в основном на версии Joomla 1.6/1.7/2.5. А там, где это уместно, подчеркиваются изменения, внесенные после версии 1.5.

Читатели, которым требуется узнать больше о Joomla и разработке расширений

В этой книге подробно рассматривается не только внутренний механизм работы Joomla, но и причины, по которым он был сконструирован именно так, а не иначе. В таком крупном и сложном пакете программ, как Joomla, был принят целый ряд проектных решений, имеющих важные последствия для разработки. Поэтому разработчикам очень важно иметь ясное представление о внутреннем механизме работы и передовом опыте применения Joomla в практике разработки веб-приложений, чтобы создавать расширения, с наибольшей выгодой использующие функциональные возможности интегрированной среды и архитектуры Joomla, а впоследствии — без труда вносить в них изменения и сопровождать.

Вопросы, не рассматриваемые в книге

В этой книге не рассматриваются шаблоны Joomla и общие вопросы проектирования. В ней также не поясняется, как пользоваться Joomla. Все эти вопросы подробно излагаются в другой литературе по Joomla.

Разработчики, работающие с Joomla, пользуются самыми разными инструментальными средствами, включая PHP, SQL, XHTML, CSS и JavaScript. Но в этой книге основное внимание уделяется написанию кода PHP, исходя из того, что у читателя нет особого опыта программирования на PHP или SQL. И поэтому приводимые примеры кода поясняются в достаточной для понимания степени. Тем не менее настоящая книга не может служить учебным пособием по PHP или SQL, и там, где это уместно, делаются ссылки на дополнительные ресурсы и сведения по рассматриваемым вопросам.

Структура книги

Книга организована по принципу "от простого — к сложному". Если у вас имеется опыт разработки веб-приложений в Joomla, можете пропустить первые две главы и начать чтение с главы 3. В противном случае вам придется начать чтение книги с самого начала, хотя сведения во врезках, где рассматриваются более сложные вопросы проектирования, вы можете пропустить.

Данной книгой можно также пользоваться как справочным пособием. Так, если вы хотели бы начать разработку с подключаемого модуля, перейдите к главе 5, обращаясь по мере необходимости к материалу предыдущих глав, если у вас недостаточно знаний и опыта.

В этой книге имеется также целый ряд врезок с дополнительными сведениями, включая рассмотрение внутреннего механизма работы Joomla, основных вопросов безопасности и прочих тем, выходящих за основные рамки книги. Поэтому можете читать врезки, исходя из собственных интересов. Кроме того, к врезкам удобно возвращаться в дальнейшем по мере надобности.

Каждый из основных видов разработки иллюстрируется отдельным учебным примером, выполняемым в пошаговом режиме. Авторы уверены, что удачный практический пример — лучший способ усвоить любой теоретический материал. Каждый этап выполнения учебного примера подробно поясняется, чтобы стало понятно, что и зачем делается на данном этапе.

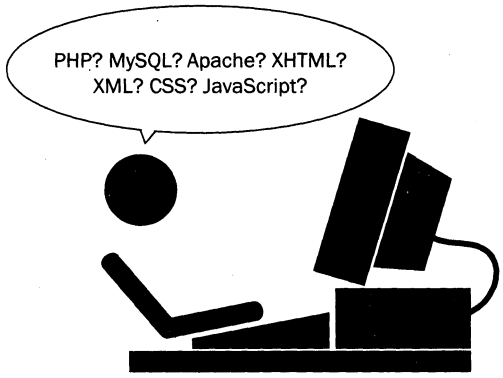
Трудности веб-разработки — нужно слишком много знать!

Один из самых сложных аспектов веб-разработки (особенно для начинающих) состоит в том, что нужно разбираться в самых разных вопросах — хотя бы в общих чертах. Например, при разработке типичного проекта в Joomla зачастую приходится иметь дело с исходным кодом PHP и запросами SQL, нередко — с разметкой документов в формате HTML и XML, а иногда — с вложенными таблицами стилей CSS и сценариями JavaScript. Кроме того, для организации рабочей среды на своем ПК нужно установить и настроить веб-сервер вроде Apache или Microsoft Internet Information Services (IIS), а также интерпретатор PHP и базу данных MySQL.

Как видите, для веб-разработки нужно знать немало, а ведь мы еще не упомянули о Joomla! Но все подобные вопросы слишком обширны, чтобы охватить их в одной книге, да и разбираться в каждом из них досконально вряд ли кому-нибудь вообще под силу.

Правда, для написания программ для Joomla совсем *не* обязательно быть специалистом во всех упомянутых выше областях веб-разработки. Но в то же время нужно в достаточной степени понимать назначение каждой из них в отдельности и ясно представлять себе их взаимодействие в комплексе.

В этой книге не предполагается, что читатель глубоко разбирается во всех вопросах веб-разработки. Все, что нужно знать по любому из рассматриваемых здесь вопросов, поясняется по ходу изложения материала. Но ради сохранения разумных рамок книги при рассмотрении отдельных тем сначала даются самые основные сведения, необходимые для уяснения излагаемого материала, а затем приводится перечень ссылок на дополнительные ресурсы для углубленного изучения данной темы на тот случай, если в этом возникнет потребность.



Нововведения в версии Joomla 2.5

Версия Joomla 1.6 была выпущена в январе 2011 года. В нее был внесен целый ряд изменений по сравнению с версией 1.5. Начиная с версии 1.6 разработчики проекта Joomla взяли на себя обязательство выпускать новую версию через каждые 6 месяцев, а версию с долгосрочной поддержкой (LTS) — через каждые 18 месяцев. В итоге версия 1.7 появилась в июле 2011 года, а версия 2.5 — в январе 2012 года.

Почему же в нумерации версий произошел столь резкий переход от 1.7 к 2.5? Это объясняется тем, что все версии с долгосрочной поддержкой должны нумероваться как X.5, где X — основная версия. Так, версия 1.5 является выпуском с долгосрочной поддержкой; версия 2.5 — выпуском с долгосрочной поддержкой для последовательного ряда версий 1.6/1.7/2.5; а версия, которую предполагается выпустить в июле 2013 года, будет следующим выпуском с долгосрочной поддержкой после версий 3.0 (в июле 2012 года) и 3.1 (в январе 2013 года).

В этой книге рассматривается версия Joomla 2.5. А поскольку основная цель книги — помочь разработчикам благополучно перейти от версии 1.5 к версии 2.5, в ней особо подчеркивается, чем последняя отличается от первой. Основные изменения в сторону усовершенствования произошли в версии 1.6. А в версии 1.7 были внесены лишь незначительные изменения и внедрены новые, хотя малозначительные средства, тогда как в версии 2.5 — дополнительные средства. Ниже вкратце рассматриваются наиболее важные среди новых средств.

Система списков контроля доступа

В версии 1.6 была внедрена система списков контроля доступа (ACL), позволяющая администраторам сайтов точно определять те действия, которые группам пользователей разрешается выполнять во внешней (пользовательской) и внутренней (административной) частях Joomla. Это очень эффективная и гибкая система, к которой могут легко обращаться сторонние разработчики. В этой книге поясняется, каким образом данная система действует и что нужно знать, чтобы воспользоваться ею в полной мере.

Определяемые пользователем уровни категорий (и больше никаких разделов)

В предыдущих версиях Joomla для статей имелись два фиксированных уровня: *раздел* и *категория*. А в версии 1.6 разделы были упразднены, и вместо них появилась возможность создавать собственную структуру категорий. Так, можно построить простую, одноуровневую структуру категорий и в то же время более сложную структуру, состоящую из подчиненных категорий до любого (но в разумных пределах) уровня глубины. Благодаря этому стало возможным создание как простых, так и сложных структур категорий, чего не допускалось прежде. И это упрощает разработчикам задачу добавления данного свойства в их собственные расширения.

Класс `JForm`

В версии 1.5 можно было легко создавать экраны для настройки параметров с помощью класса `JParameter`. А в версии 1.6 ему на смену пришел класс `JForm`, упрощающий создание эффективных форм для приложений. Класс `JForm` благополучно сочетает в себе удобство и простоту применения в тех случаях, когда в Joomla требуется создать заполняемые данными формы. С этой целью все административные экраны были переписаны для версии 1.6 Joomla с помощью класса `JForm`.

Одним из последствий подобных нововведений явилось изменение формата XML-файлов для расширений. Более подробно этот вопрос рассматривается в главах, посвященных написанию расширений.

Класс `JTableNested`

Класс `JTableNested` является новым базовым классом для создания категорий, меню и таблиц с вложенной многоуровневой структурой элементов. Он предоставляет прикладной программный интерфейс (API), упрощающий разработчикам создание в прикладном коде таблиц на основе вложенных множеств.

Класс `JDatabaseQuery`

Это новый класс, упрощающий составление сложных запросов SQL в Joomla. Он предоставляет интерфейс API для построения запросов SQL в логичном порядке, исходя из логической структуры запроса. Благодаря этому значительно упрощается написание и сопровождение сложных запросов SQL. Вам, возможно, и не придется пользоваться этим новым классом при составлении запросов SQL, но, ознакомившись с ним поближе, вы вряд ли станете отрицать, что с его помощью обработка запросов SQL в Joomla оказывается более эффективной.

Версия PHP 5.2

Версия Joomla 1.5 должна была непременно быть совместимой с версией PHP 4.0. Это накладывало определенные ограничения на использование улучшений, внесенных в версии 5.0 и 5.2 языка PHP в отношении объектно-ориентированного программирования (ООП).

Начиная с версии 1.6 для работы в Joomla требуется версия PHP 5.2 или еще более новая версия этого языка. Это позволяет применять статические и абстрактные классы в версии Joomla 1.6. Кроме того, в версии PHP 5.2 по умолчанию все объекты передаются по ссылке, а это означает, что оператор `&=` (присваивания по ссылке) больше не требуется, за исключением особых случаев.

Обновленная версия PHP допускает также применение в Joomla собственного класса `SimpleXML` для синтаксического анализа XML-файлов и собственного класса `DateTime`. В связи с этим соответствующие упрощения и видоизменения были внесены в классы `JXMLElement` и `JDate`.

Версия MySQL 5.0.4

Для работы в Joomla, начиная с версии 1.6, требуется также версия MySQL 5.0.4 или еще более новая версия этой базы данных. В данную версию MySQL был внесен целый ряд усовершенствований, в том числе хранимые процедуры, триггеры, представления, а также ряд усовершенствований, направленных на повышение производительности. Это позволяет также отображать таблицы с крупными столбцами типа `varchar`.

Формат файлов языковой поддержки

В версии Joomla 1.6 было внесено существенное изменение в формат файлов языковой поддержки. Раньше в Joomla для этой цели применялся оригинальный формат. А в версии 1.6 он был заменен на стандартный для PHP формат файлов с расширением `.ini`. Благодаря этому появилась возможность использовать в Joomla стандартную для PHP команду `parse_ini_file`, что оказывается много быстрее и проще, чем старый оригинальный метод. Но для этого придется переформатировать файлы языковой поддержки по новому стандарту.

Упрощенная процедура обновления

Версия 2.5 допускает автоматическое обновление веб-сайта, построенного на основе Joomla. Администратор сайта уведомляется всякий раз, когда доступно обновление как базовых файлов Joomla, так и любых расширений, применяемых на сайте (при условии, что разработчик расширения поддерживает такую возможность). Для обновления веб-сайта достаточно щелкнуть на кнопке Update (Обновить). Инструкции по установке подобной возможности для обновления расширений можно найти, перейдя по адресу `http://joomlaprogrammingbook.com` и щелкнув на ссылке `Implementing One-Click Update` (Реализация обновления одним щелчком кнопкой мыши).

Усовершенствования шаблона MVC

Шаблон проектирования MVC (модель–представление–контроллер) был усовершенствован в версии 1.6. К числу внесенных усовершенствований относится более

согласованное применение данного шаблона в административной части веб-сайта и более рациональное повторное использование кода благодаря его переносу в классы стандартной библиотеки там, где это возможно. Подробнее реализация шаблона MVC в Joomla рассматривается в главах 7–10.

Поддержка других баз данных

В версии 2.5 внедрена поддержка других баз данных, кроме MySQL. Этот процесс начат с Microsoft SQL Server. А поддержка PostgreSQL находится на стадии разработки и ожидается доступной в последующих версиях Joomla.

Усовершенствованный поиск

В версии 2.5 внедрена также возможность интеллектуального поиска (Smart Search). Это совершенно новый поисковый механизм, значительно улучшающий качество и точность текстового поиска по всему содержимому веб-сайта, построенного на основе Joomla.

Веб-сайт, сопровождающий книгу

Авторы книги организовали веб-сайт по адресу `http://joomlaprogrammingbook.com`, где можно найти дополнительные сведения о программировании для Joomla. Здесь же находятся разделенные по главам архивные файлы с примерами исходного кода, приведенными в тексте книги.

Этот веб-сайт будет регулярно обновляться новой информацией о версиях Joomla и коррективами, внесенными в печатное издание книги.

Добро пожаловать в сообщество разработчиков Joomla

Разработчики Joomla имеют разную техническую подготовку и опыт проектирования программного обеспечения. Многие из них начинали с HTML-сайтов и узнали немало о веб-разработке, работая с Joomla. Одни из них имеют высшее образование в области вычислительной техники, другие — специальную подготовку по веб-дизайну, а третьи научились пользоваться Joomla по первоисточникам и в совместной работе с другими разработчиками.

Сообщество разработчиков Joomla стремится к открытости и приветствует новых членов, в том числе и тех, у кого мало опыта программирования или недостаточно технического образования. Для изучения различных аспектов разработки веб-приложений в Joomla и поиска ответов на насущные вопросы имеется немало интернет-ресурсов. Поэтому ниже перечислены лишь два самых важных из них.

- `http://developer.joomla.org`. Этот сайт посвящен исключительно разработке в Joomla. На нем размещаются сведения о текущем состоянии проекта Joomla! и последние новости по разработке. Здесь же приводятся ссылки на группы новостей в Google, где также обсуждаются вопросы разработки веб-приложений в Joomla.

- `http://docs.joomla.org/Developers`. Этот сайт служит в качестве отправной точки для поиска всего, что связано с разработкой в Joomla. Однако этот сайт построен по принципу вики и поддерживается самим сообществом разработчиков. Всякий его посетитель может зарегистрироваться и обновить или добавить информацию в избранные редактируемые статьи вики.

Авторы надеются, что эта книга облегчит читателям усвоение особенностей разработки прикладных программ для Joomla независимо от их предыдущего опыта программирования.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши электронные адреса:

E-mail: `info@williamspublishing.com`

WWW: `http://www.williamspublishing.com`

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д.43, стр. 1

в Украине: 03150, Киев, а/я 152

1

Основы разработки в Joomla!

В этой главе в общих чертах рассматриваются различные способы расширения и специальной настройки функциональных возможностей Joomla!. Но прежде необходимо хотя бы вкратце описать разнообразные программы, применяемые для построения и выполнения проекта в Joomla.

Разрабатывать в Joomla совсем не трудно!

Если вы только начинаете осваивать Joomla в частности и веб-разработку в целом, то вас могут привести в замешательство многочисленные сокращения, специальные термины и новые понятия. Но, проявив немного настойчивости и обратившись за помощью к первоисточникам, вы сможете преуспеть в написании кода, чтобы заставить Joomla работать именно так, как вам нужно. В Joomla изначально была заложена возможность видоизменения и расширения, и многие пользователи даже с ограниченной технической подготовкой успешно научились писать программы для Joomla.

В этой книге поясняется каждое понятие и раскрывается новая тема по мере изложения. Авторы изначально не предполагают, что читатель уже знаком с Joomla или веб-разботкой. Поэтому там, где это необходимо, будут даны ссылки на дополнительные ресурсы для восполнения пробела в знании рассматриваемого предмета или более углубленного его изучения.

Техническая среда Joomla

Начинающих осваивать разработку динамических веб-сайтов нередко сбивает с толку поразительное разнообразие сокращений и специальных терминов, применяемых в данной области. Так, в Joomla используются серверы типа Apache или Microsoft Internet Information Services (IIS); язык программирования PHP; база

данных MySQL; язык программирования JavaScript; документы формата HTML, XHTML и XML; а также вложенные таблицы стилей CSS.

Начнем с сервера. Сервер — это компьютер, на котором находятся файлы программ Joomla и выполняется большая (но не вся) часть программного кода Joomla. Для нормальной работы веб-сайта, построенного на Joomla, на сервере должны быть запущены на выполнение три следующих программных компонента:

- веб-сервер (Apache или Microsoft IIS);
- язык программирования PHP;
- база данных (обычно MySQL).

Веб-сервер — это программное обеспечение взаимодействия сервера и браузера. Это своего рода мост между внешним миром и локальными папками на сервере. Когда браузер запрашивает URL (т.е. информационный ресурс по указанному веб-адресу), веб-сервер знает, какой именно каталог файлов сопоставить с этим URL, и находит нужный файл (например, `index.php`) для выполнения.

PHP — это основной язык программирования, на котором написан код Joomla. Без сомнения, PHP является самым распространенным языком программирования для разработки динамических веб-сайтов. Программа, написанная на этом языке, взаимодействует с программным обеспечением веб-сервера и создает XHTML-страницы в динамическом режиме. Для решения большинства задач разработки в Joomla программирование выполняется на языке PHP.

Сервер как понятие и как машина

Употребляя понятие *сервер* в контексте веб-разработки, мы имеем в виду программное обеспечение, предоставляющее браузеру код HTML для выполнения. На рабочем веб-сайте программное обеспечение веб-сервера обычно выполняется на удаленном компьютере компании, предоставляющей услуги веб-хостинга (т.е. размещения веб-сайтов). Но даже если мы работаем с Joomla в локальном режиме, что часто делается для целей разработки, отладки и тестирования веб-сайтов, то и тогда имеем дело с сервером. В этом случае в качестве сервера служит программное обеспечение (Apache или IIS), работающее на нашем ПК.

С технической точки зрения обе среды действуют одинаково. Сервер и браузер всегда связываются по протоколу TCP/IP. Единственное отличие заключается в том, что в среде с удаленным сервером данные переносятся туда и обратно в пакетах TCP/IP на гораздо большее расстояние.

MySQL — это база данных, применяемая на большинстве сайтов, построенных на Joomla. Именно эта база данных чаще всего выбирается для веб-приложений. В базе данных веб-сайта на Joomla хранится практически вся информация о самом сайте. Например, всякий раз, когда вы добавляете статью, меню или модуль в структуру веб-сайта, этот компонент сохраняется в базе данных.

Когда вы пользуетесь веб-сайтом, построенным на Joomla, или иными средствами, то в окне своего браузера, по существу, наблюдаете XHTML-документ. Сокращение XHTML обозначает расширяемый язык гипертекстовой разметки, который является обновленной

версией HTML — языка гипертекстовой разметки, исходно применявшегося в веб-разработке. Для разработки веб-сайтов в Joomla требуются хотя бы элементарные знания языка XHTML. Ведь конечным результатом всей обработки, выполняемой программой Joomla, является отображение XHTML-документа в окне браузера. Следует, однако, иметь в виду, что далее в книге термином *HTML* будут обозначаться оба языка, HTML и XHTML, а также их форматы и документы.

Сокращение CSS обозначает вложенные таблицы стилей. Код CSS осуществляет стилевое оформление XHTML-документов для их отображении в окне браузера должным образом. В частности, таблицы CSS определяют размер, тип, начертание, цвет шрифта и расстановку пробелов в тексте и все остальные аспекты форматирования данных, отображаемых в окне браузера.

Все сведения о том, как должен быть отображен веб-сайт, включая таблицы CSS и расположение содержимого на веб-странице, содержатся в шаблоне Joomla. Такая конструкция позволяет полностью отделить содержимое сайта от его внешнего вида. Благодаря этому внешний вид веб-сайта на Joomla можно совершенно изменить динамически, заменив лишь шаблон.

JavaScript — это язык написания сценариев, т.е. специальных программ, выполняющихся в браузере и управляющих отображением содержимого при просмотре веб-страниц. Обозначается буквой "J" в сокращении названия технологии AJAX (асинхронный JavaScript и XML). Благодаря сценариям на языке JavaScript веб-сайты воспринимаются как настольные прикладные программы. Так, в Joomla сценарии JavaScript организуют автоматическое всплывание подсказки при наведении курсора мыши на поле, а также выделение поля в форме, если оно заполнено неверно. Сценарий JavaScript выполняется на стороне клиента и поэтому может реагировать на события, не взаимодействуя с веб-сервером. Благодаря этому изменения в окне браузера происходят сразу, не вынуждая пользователя щелкать на кнопке Submit (Отправить).

Требования к программированию для Joomla

Программирование для Joomla заключается главным образом в написании кода на языке PHP. Информация для веб-сайта на Joomla хранится в базе данных, и поэтому часть этого кода PHP взаимодействует с базой данных. Как правило, такое взаимодействие осуществляется в форме запросов SQL. Запросы базы данных служат для сохранения и извлечения из нее данных. Поэтому разработчикам Joomla нужно знать и уметь составлять запросы SQL для организации нормального взаимодействия с базой данных.

Частично программирование для Joomla заключается и в работе с XML или HTML-документами. Например, параметры форм обычно хранятся в XML-файлах. Те части программы, которые отвечают за вывод информации в окно браузера (так называемых представлений или компоновок), обычно содержат смешанный код PHP и HTML. Поэтому разработчикам Joomla нужно знать хотя бы основы HTML.

Одни разработчики в Joomla занимаются в основном компоновкой и внешним видом веб-сайта. И для этого им требуются более основательные знания HTML и CSS. Большая часть подобной работы выполняется на стадии разработки шаблона. Но, как упоминалось ранее, разработка шаблонов в этой книге не рассматривается.

Другие разработчики в Joomla занимаются правильным отображением информации на веб-странице. И хотя для этого не нужно быть знатоком HTML или CSS, общие представление об этих программных средствах веб-разботки совсем не помешает.

Особая роль принадлежит языку JavaScript. Сценарии JavaScript взаимодействуют с программами PHP, а также с кодом HTML и CSS. Но для разработки в Joomla особых знаний языка JavaScript и тем более программирования сценариев на нем не требуется. Более подробно роль языка JavaScript в Joomla рассматривается в главе 12.

Разработчикам не нужно особенно много знать о веб-сервере (Apache или IIS). При развертывании веб-сайта в Интернете особое внимание уделяется его безопасности. В значительной степени безопасность сайта зависит от того хоста (т.е. веб-узла), на котором размещается сайт, а также от настройки программного обеспечения веб-сервера и базы данных на хосте. Но, как правило, это не оказывает особого влияния на само написание программ для Joomla.

Итак, подводя итог, следует сказать, что на веб-сайте, построенном на Joomla, используется целый ряд программ и типов файлов, обеспечивающих работоспособность сайта. Большая часть разработки в Joomla связана с написанием кода PHP, чему и уделяется основное внимание в этой книге. Такой код отчасти содержит запросы базы данных. Более подробно вопросы управления базой данных рассматриваются в главе 11. В другой части кода PHP организуется вывод HTML-документа в окно браузера. При создании окончательного варианта вывода будет разъяснено, каким образом организуется изящное взаимодействие кода PHP и HTML.

Большую часть разработки в Joomla можно, конечно, выполнить, не зная вообще ничего о JavaScript. Тем не менее эта важная тема не будет оставлена без внимания и подробно рассматривается в главе 13.

Способы расширения Joomla

Как упоминалось ранее, в Joomla изначально была заложена возможность расширения. Что же это означает на практике? Расширение, по существу, означает приспособление Joomla под свои нужды. Это может быть сделано в виде изменения режима работы какого-нибудь компонента или добавления совершенно новой функциональной возможности. В этом разделе вкратце рассматриваются различные способы расширения Joomla.

Открытость кода означает полный контроль над ним

Joomla является свободно доступным программным обеспечением с открытым кодом (FOSS), и поэтому его исходный код включается в обычный загружаемый пакет. Всякий, способный построить веб-сайт на Joomla, может при желании отредактировать содержимое любых исходных файлов или изменить порядок выполнения программы.

Определим два термина, которыми вам предстоит в дальнейшем пользоваться при обсуждении с коллегами видоизменений программ в Joomla.

- *Базовый код* — исходный код, распространяемый в наборе файлов, получаемых при загрузке и установке Joomla (например, в виде архивного файла).
- *Доработка базового кода* — внесение изменений непосредственно в базовый исходный код.

В связи с тем что Joomla является свободно доступным программным обеспечением с открытым кодом, вы имеете полное право доработать содержимое любого исходного файла в данной системе. Но делать этого не рекомендуется по двум простым причинам. Во-первых, если вы внесете изменения в файлы базового исходного кода, то уже не сможете полностью рассчитывать на целостность системы. Так, если в работе системы возникнет сбой, вам вряд ли удастся выяснить, является ли тому причиной Joomla или результат ваших изменений. И во-вторых, обновление веб-сайта существенно усложнится, что еще хуже. Как правило, версии, сопровождаемые в рамках проекта Joomla, выпускаются с регулярными обновлениями. Так, при первоначальном запуске веб-сайта может быть использована версия Joomla 2.5.1, а впоследствии — вновь выпущенная версия 2.5.2.

Архивный файл версии 2.5.2 будет содержать полную копию каждого файла, измененного с момента выпуска предыдущей версии Joomla, и поэтому при обновлении старые файлы будут заменены. Если же вы внесли изменения в некоторые из этих файлов, то в результате обновления утратите все эти изменения. А если вы не обновите эти файлы, то вам будут недоступны исправления программных ошибок, обнаруженных в текущей версии, и прочие изменения. И тогда остается лишь один выход: повторить внесенные ранее изменения в файлах новой версии или выявить те изменения, которые произошли в новой версии по сравнению со старой, а затем внести их аналогичным образом в видоизмененные вами файлы. Возможно, это нетрудно будет сделать, если произведенные вами доработки незначительны. Но в целом это усложняет весь процесс обновления веб-сайта.

Правда, доработка файлов основного исходного кода для нормальной работы Joomla требуется крайне редко. В последующих разделах поясняются различные способы видоизменения Joomla без необходимости вносить коррективы в файлы основного исходного кода.

Изменения на веб-странице с помощью файлов замещения

К числу изменений, которые чаще всего приходится вносить на сайте, относится видоизменение того, что должно отображаться на веб-странице. Если у вас имеется некоторый опыт применения Joomla, то вам должно быть известно, что большинство модулей, компонентов и подключаемых модулей предоставляют параметры, с помощью которых можно управлять различными режимами работы этих расширений. Так, если на странице отображается статья, то с помощью соответствующего параметра вы можете настроить режим показа или сокрытия автора и даты публикации данной статьи. Для внесения подобных изменений никакого программирования не требуется.

Но ведь сколько людей, столько и мнений по поводу отображения информации на странице, и поэтому не имеет никакого практического смысла пытаться настраивать параметры под каждую возможность вывода информации. Обратимся к простому примеру. На рис. 1.1 показано, каким образом в Joomla выглядит стандартная компоновка списка статей.

Допустим, нам нужно показывать автора статьи в первом столбце, а название статьи — во втором. Но параметр для настройки отображения столбцов в нужном порядке отсутствует. Что же делать? Просто скопировать один файл, вырезать и вставить несколько строк кода, изменив таким образом порядок вывода информации, как показано на рис. 1.2.

Display # 5

Title	Author	Hits
Beginners	Written by Joomla!	1
Getting Help	Written by Joomla!	16
Getting Started	Written by Joomla!	6
Joomla!	Written by Joomla!	0
Parameters	Written by Joomla!	1

Рис. 1.1. Стандартная компоновка статей в одной категории

Display # 5

Author	Title	Hits
Written by Joomla!	Beginners	1
Written by Joomla!	Getting Help	16
Written by Joomla!	Getting Started	6
Written by Joomla!	Joomla!	0
Written by Joomla!	Parameters	1

Рис. 1.2. Компоновка статей, видоизмененная с помощью файла замещения шаблона

Подобное изменение было сделано в виде замещения, а это означает, что мы не дорабатывали ни один из файлов базового кода. И поэтому нам не нужно особенно беспокоиться о том, что файлы будут перезаписаны при переходе к новой версии Joomla.

Начиная с версии 1.6 область применения файлов замещения была значительно расширена. Файлы замещения шаблона поддерживаются по-прежнему. Но вместе с тем появилась возможность замещать файлы языковой поддержки, пункты меню и модули независимо от используемого шаблона. Подробнее о подобных замещениях речь пойдет в главе 4.

Расширения

Расширения относятся к самым заметным свойствам любого веб-сайта на Joomla. Термином *расширение* обычно принято обозначать вспомогательное программное обеспечение, работающее вместе с Joomla и дополняющее функциональные возможности веб-сайта. В рамках проекта Joomla поддерживается веб-сайт Joomla Extensions Directory (JED — Каталог расширений Joomla; `http://extensions.joomla.org`), на котором перечислены многочисленные расширения, доступные для сообщества разработчиков.

В программировании термин *расширение* имеет несколько иное значение. В данном контексте он обозначает расширение любого компонента, встроенного или подключаемого модуля, языковой поддержки или шаблона. Расширения, включаемые в стандартный загружаемый пакет Joomla, называются *базовыми*, а расширения из других источников (например, JED) — *сторонними*.

Важно понимать, что во время работы Joomla не отличает базовые расширения от сторонних. Именно поэтому сторонние расширения оказываются полностью интегрированными в пакет программ, где они трактуются точно так же, как и базовые расширения.

Тем, кто пользуется веб-сайтом на Joomla, включающим в себя грамотно написанные сторонние расширения, зачастую очень трудно провести четкую границу там, где оканчивается базовый пакет и начинается расширение. Это конструктивная особенность Joomla.

Подключаемые модули

Вероятно, подключаемые модули — самое секретное оружие разработки в Joomla. Зачастую подключаемые модули позволяют реализовать те функции веб-сайта, которые трудно, а то и просто невозможно воплотить в жизнь иначе. Но поскольку они, как правило, малозаметны, то их очень легко упустить из виду в качестве оптимального решения конкретной задачи веб-разработки.

Для того чтобы лучше понять истинное назначение подключаемых модулей, достаточно представить ход выполнения программы в виде ленты конвейера или сборочной линии. В различных точках данного процесса наступают отдельные события. На рис. 1.3 показана блок-схема программы для сохранения статьи в базе данных.

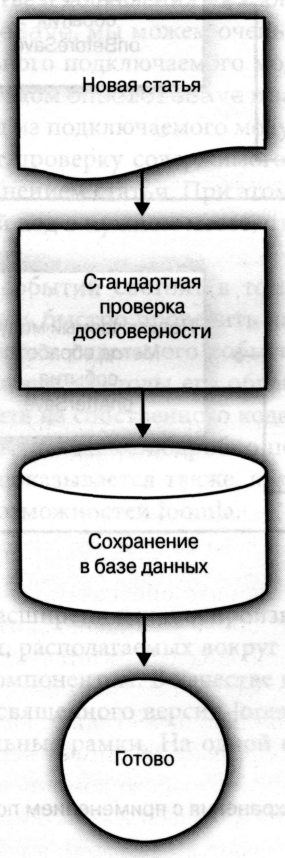


Рис. 1.3. Простой процесс сохранения

На первой стадии рассматриваемого здесь процесса пользователь вводит статью и щелкает на кнопке Save (Сохранить). На второй стадии в Joomla выполняется проверка достоверности введенных данных из статьи. Если эта проверка проходит успешно, статья сохраняется в базе данных. И на последней стадии показывается, что все готово и процесс завершен. В системе обработки событий, применяемой в Joomla, данный процесс выглядит посложнее, но выполняется намного эффективнее, как показано на рис. 1.4.

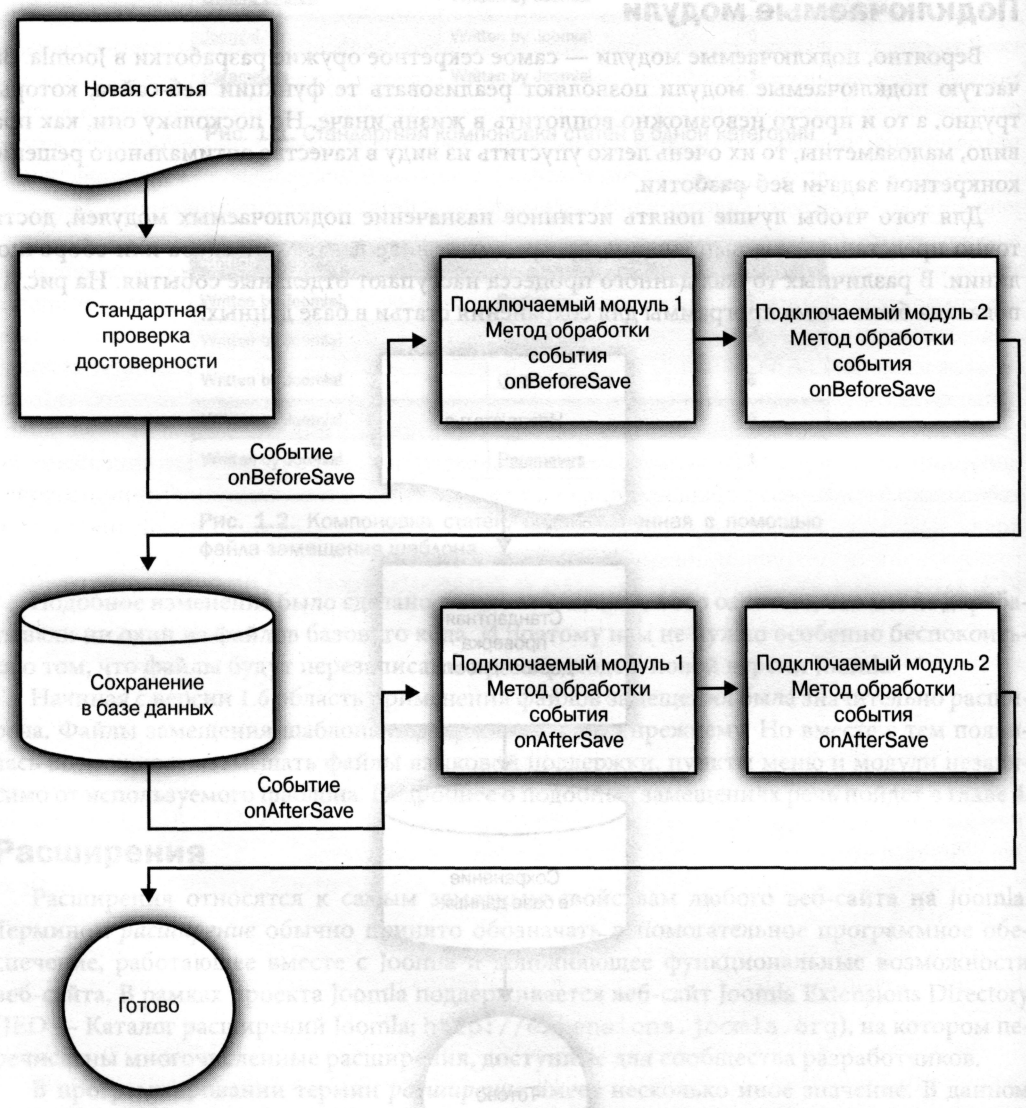


Рис. 1.4. Процесс сохранения с применением подключаемых модулей

Основные стадии процесса, следующие по блок-схеме сверху вниз, остаются теми же, что и прежде, но в данном случае они дополняются двумя событиями. Первое событие, называемое `onBeforeSave`, наступает после проверки достоверности данных из статьи, но перед их сохранением в базе данных. А второе событие, называемое `onAfterSave`, наступает вслед за успешным сохранением в базе данных, но перед завершением процесса. Эти события прерывают обычный ход выполнения программы, позволяя организовать их специальную обработку. По завершении этой обработки обычный ход выполнения программы возобновляется.

Как следует из рис. 1.4, в Joomla обнаруживаются два подключаемых программных модуля с методами `onBeforeSave` и `onAfterSave` для обработки упомянутых выше событий. Следует, однако, иметь в виду, что в зависимости от реализации Joomla количество модулей, подключаемых и выполняемых для обработки событий, может находиться в пределах от 0 до 10. Всякий раз, когда сохраняется статья, в Joomla сначала проверяются любые методы, которые следует выполнить, а затем они выполняются по очереди.

Обратимся к конкретному примеру. Допустим, нам требуется проверить каждую статью, публикуемую на сайте, и убедиться в том, что в ней имеются ключевые слова, определенные в метаданных. Допустим также, что мы уже написали программу для поиска определенных слов в каждой статье и добавления их к ключевым словам.

Используя событие `onBeforeSave`, мы можем очень просто реализовать описанную выше функцию в виде специального подключаемого модуля. Для этого достаточно создать подключаемый модуль с методом `onBeforeSave` и запустить нашу программу на выполнение. В данном случае метод из подключаемого модуля принимает статью в качестве своего параметра, что упрощает проверку содержимого статьи и обновление столбца с ключевыми словами перед сохранением статьи. При этом мы не изменяем файлы базового кода, а лишь добавляем новый код в нужном месте, чтобы он был обнаружен в Joomla и выполнен в подходящий момент.

Замечательная особенность событий состоит в том, что они всегда присутствуют, но не мешают. Joomla может очень быстро проверить наличие любых методов, которые требуется выполнить для обработки отдельного события. Событие считается инициированным, если программа проверяет методы его обработки и, обнаружив, выполняет их. События можно инициировать из собственного кода, а по мере необходимости даже сформировать специальные события. Более подробно подключаемые модули и события рассматриваются в главе 5, где показывается также, как выгодно пользоваться ими для расширения функциональных возможностей Joomla.

Модули

В Joomla модули являются расширениями, воспроизводимыми на веб-странице в небольших прямоугольных рамках, располагаемых вокруг основного содержимого страницы, которое обычно является компонентом. В качестве примера на рис. 1.5 показана начальная страница веб-сайта, посвященного версии Joomla 2.5. Модули на этой странице заключены в черные прямоугольные рамки. На одной странице можно отобразить несколько модулей.

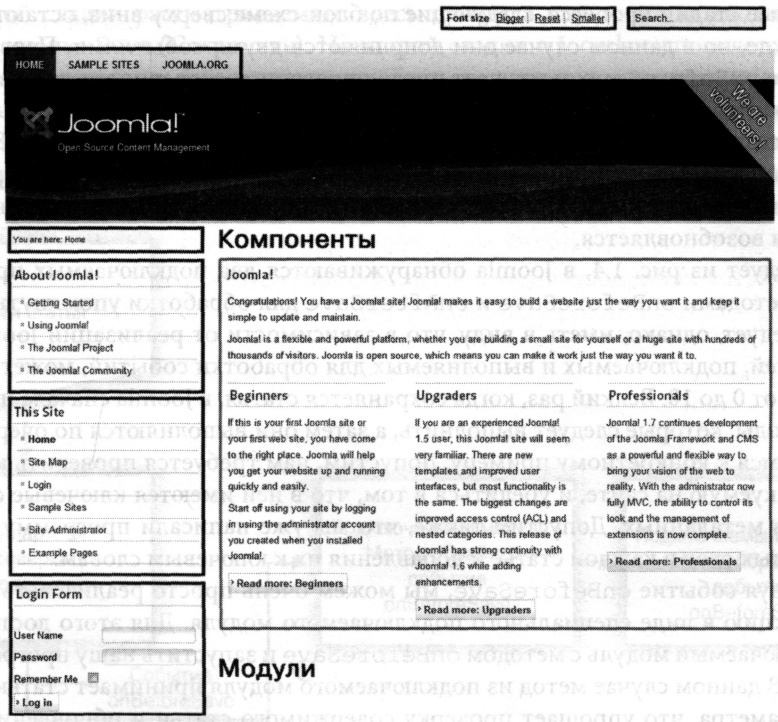


Рис. 1.5. Пример начальной страницы с одним компонентом и несколькими модулями

Как правило, модули являются простыми расширениями. Модуль может быть привязан к компоненту. Например, модуль `Related Items` (Связанные предметы) привязан к представлению `Single Article` (Единственная статья) компонента `Articles` (Статьи) и отображает список статей с ключевыми словами, совпадающими с аналогичными словами в статье из представления. В других случаях модуль отображает ту же самую информацию независимо от компонента, присутствующего на странице.

Все меню в Joomla являются модулями, и в базовый код Joomla включено немало других типов модулей. Подробнее о базовых модулях речь пойдет в главе 3. Как правило, модуль выбирается для использования в том случае, если информацию требуется отобразить на нескольких страницах и она не должна привлекать основное внимание на данной странице. Более подробно разработка модулей рассматривается в главе 6.

Компоненты

Как правило, компонент сосредоточивает основное внимание на веб-странице. В примере, приведенном на рис. 1.5, в области содержимого, выделенной в серой рамке, отображается пункт меню Featured Articles — один из пунктов меню, включенных в компонент статей. В компоненты входит один или несколько типов пунктов меню, выбираемых при создании или редактировании пункта меню. Каждому типу пункта меню соответствует представление компонента. Например, компонент статей предоставляет на выбор семь

возможных типов пунктов меню или представлений: Archived Articles (Архивные статьи), Single Article (Одиночная статья), List All Categories (Список всех категорий), Category Blog (Блог по категории), Category List (Список категорий), Featured Articles (Избранные статьи) и Create Article (Создать статью).

К примерам базовых компонентов Joomla относятся статьи (называемые также содержимым, или `com_content`), крупные заголовки (называемые также баннерами, или `com_banners`) и контактная информация (называемая также контактами, или `com_contact`). Более подробно базовые компоненты рассматриваются в главе 3. На рис. 1.6 приведены все типы пунктов меню, доступные для базовых компонентов.

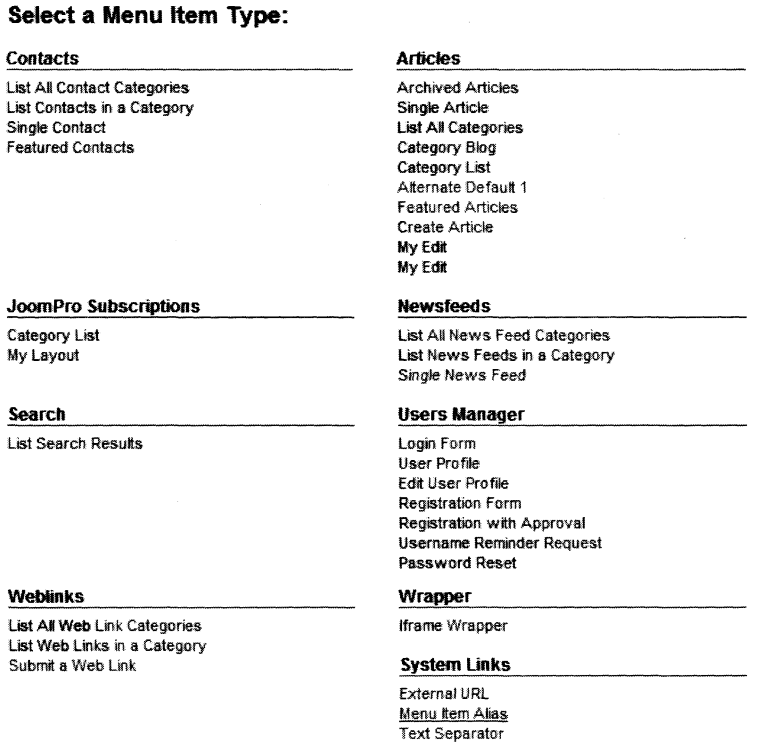


Рис. 1.6. Типы пунктов меню, доступные для основных компонентов в версии Joomla 2.5

Компоненты в Joomla написаны с использованием шаблона проектирования MVC (модель–представление–контроллер). Благодаря этому упрощается сопровождение программ в течение срока их эксплуатации и допускается разделение на отдельные задачи, связанные с подготовкой и отображением веб-страницы средствами Joomla. Кроме того, применение данного шаблона значительно упрощает разработчикам, знакомым с объектно-ориентированным программированием (ООП) и шаблонами проектирования, понимание принципа действия компонентов в Joomla.

Как правило, компоненты относятся к самому сложному типу расширений. Более подробно написание компонентов рассматривается в главах 7–10.

Языковые расширения

К сильным сторонам Joomla относится высокая степень локализации. В настоящее время осуществлена локализация Joomla на более чем 68 языков мира. Это дает пользователям возможность работать с программой Joomla на их родном языке.

Локализация Joomla реализуется с помощью языковых расширений. Языковое расширение представляет собой набор специальных файлов (с расширением `.ini`), содержащих все слова и выражения, отображаемые в пользовательском интерфейсе Joomla. Это позволяет перевести пользовательский интерфейс на любой язык мира. Покажем, как это делается.

Для отображения текста на странице, выводимой в окне браузера, в языке PHP служит команда `echo`. Рассмотрим для примера экран регистрации администратора на веб-сайте Joomla, как показано на рис. 1.7.

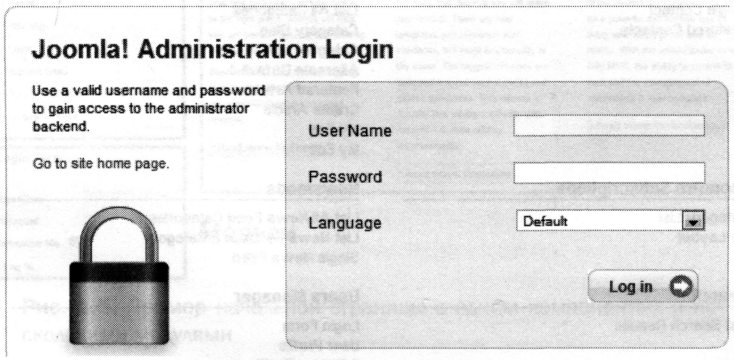


Рис. 1.7. Экран регистрации администратора

Глядя на экран, приведенный на рис. 1.7, можно предположить, что где-то в недрах кода PHP для Joomla находится команда вроде `echo 'Joomla! Administration Login'` для регистрации администрирования Joomla!. Но если мы попытаемся найти эту команду в файле исходного кода PHP, то ничего подобного нам найти в итоге не удастся.

В Joomla текст оформления пользовательского интерфейса не кодируется жестко, иначе пришлось бы вносить изменения в программный код для каждого языка мира в отдельности. Вместо этого жестко кодируются ключи в файлах языковой поддержки, а их значения читаются как конкретный текст во время выполнения. Если же мы попытаемся найти символьную строку `'Joomla! Administration Login'` в файле языковой поддержки с расширением `.ini`, то в конечном итоге обнаружим ее в файле `administrator/language/en-GB/en-GB.com_login.ini`, как показано на рис. 1.8.

Левая часть `COM_LOGIN_JOOMLA_ADMINISTRATION_LOGIN` восьмой строки кода в данном файле называется языковым ключом (или дескриптором). А в правой ее части, заключенной в кавычки и представляющей собой значение ключа, содержится конкретный текст, отображаемый средствами Joomla при выборе британского английского языка (`en-GB`).

```
en-GB.com_login.ini
 1 ; $Id: en-GB.com_login.ini 20196 2011-01-09 02:40:25Z ian $
 2 ; Joomla! Project
 3 ; Copyright (C) 2005 - 2011 Open Source Matters. All rights reserved.
 4 ; License GNU General Public License version 2 or later; see LICENSE.t
 5 ; Note : All ini files need to be saved as UTF-8
 6
 7 COM_LOGIN="Login"
 8 COM_LOGIN_JOOMLA_ADMINISTRATION_LOGIN="Joomla! Administration Login"
 9 COM_LOGIN_RETURN_TO_SITE_HOME_PAGE="Go to site home page."
10 COM_LOGIN_VALID="Use a valid username and password to gain access to t
11 COM_LOGIN_XML_DESCRIPTION="This component lets users login the site."
```

Рис. 1.8. Результаты поиска текста в файле с расширением `.ini`

Как видите, в результате поиска по данному языковому ключу (или дескриптору) в исходных файлах PHP он обнаруживается дважды в файлах `administrator/templates/bluestork/login.php` и `administrator/templates/hathor/login.php`. Ниже приведена строка кода с командой `echo`.

```
<h1><?php echo JText::_('COM_LOGIN_JOOMLA_ADMINISTRATION_LOGIN') ?></h1>
```

Вместо команды `echo` с текстом на английском языке мы можем вывести результат выполнения метода `JText::_()`. В качестве аргумента этого метода указывается языковый ключ из файла с расширением `.ini`. Метод `JText::_()` осуществляет поиск по заданному языковому ключу в файле с расширением `.ini` и возвращает совпадающий текст. Если же по какой-нибудь причине поиск по заданному языковому ключу не даст никакого результата, метод возвратит сам ключ.

Если бы было установлено и выбрано другое языковое расширение, называемое иначе языковым пакетом, то в результате аналогичного поиска появился бы текст на данном языке. В качестве примера на рис. 1.9 приведен экран регистрации на китайском языке, если установлен пакет именно этого языка.

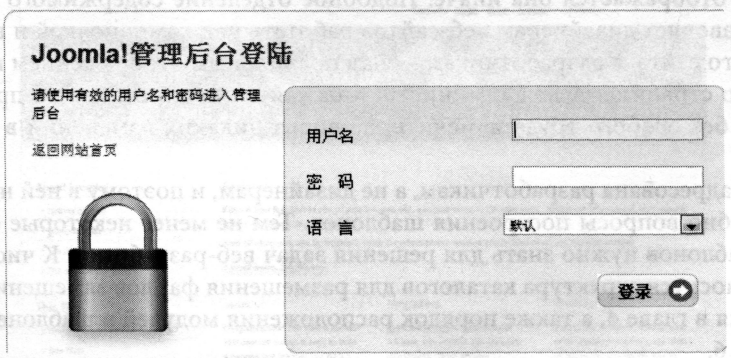


Рис. 1.9. Экран регистрации на упрощенном китайском языке

Для создания языковых расширений следует сформировать новые файлы, соответствующие каждому основному файлу с расширением `.ini`, где все дескрипторы переведены на целевой язык. Нетрудно себе представить, насколько трудоемкой оказывается

такая задача, и поэтому нет ничего удивительного в том, что самой большой в рамках проекта Joomla является группа разработчиков, занимающаяся локализацией Joomla на более чем 68 языков мира (Joomla Translation Team).

Если вы будете создавать расширение для сообщества пользователей Joomla, настоятельно рекомендуется сформировать для него файлы языковой поддержки с расширением `.ini` и воспользоваться методами из класса `JText` для вывода текста. Тем самым вы облегчите другим задачу локализации вашего расширения на другие языки мира. В главе 5 на конкретном примере будет показано, как это делается при добавлении файла языковой поддержки в расширение подключаемого модуля.

Шаблоны

И последним типом расширений в Joomla является шаблон, управляющий отображением всей информации на каждой странице веб-сайта. Основной принцип Joomla (и других систем управления информационным наполнением веб-сайтов) состоит в отделении содержимого сайта от представления этого содержимого. С этой целью процесс подготовки каждой страницы веб-сайта разделяется на отдельные стадии. Сначала в Joomla создается весь документ для каждой страницы, а затем на завершающей стадии данного процесса этот документ передается шаблону, где он, по существу, преобразуется в окончательный формат HTML, в котором он и будет направляться в дальнейшем для отображения в браузере.

Результат такого отделения содержимого от его представления проявляется явным образом на примере изменения используемого по умолчанию шаблона. Так, если перейти к пользовательской части сайта с шаблоном `Beez2`, появится его начальная страница, приведенная на рис. 1.10.

Если перейти к компоненту Template Manager (Диспетчер шаблонов) и внести изменения в используемый по умолчанию шаблон `Beez5-Default-FruitShop`, то появится начальная страница, аналогичная приведенной на рис. 1.11. Ее содержимое остается прежним, но отображается она иначе. Подобное отделение содержимого от его представления позволяет дизайнерам веб-сайтов работать над компоновкой и внешним видом отдельного сайта, а разработчикам — над правильным отображением информации на каждой его странице. А для администратора сайта это означает, что представление сайта можно без особого труда изменить, не внося никаких изменений в порядок его построения.

Эта книга адресована разработчикам, а не дизайнерам, и поэтому в ней не рассматриваются подробно вопросы построения шаблонов. Тем не менее некоторые особенности структуры шаблонов нужно знать для решения задач веб-разработки. К числу этих особенностей относится структура каталогов для размещения файлов замещения шаблонов, как поясняется в главе 4, а также порядок расположения модулей в шаблоне, как обсуждается в главе 6.



Рис. 1.10. Начальная страница веб-сайта с шаблоном Beez2

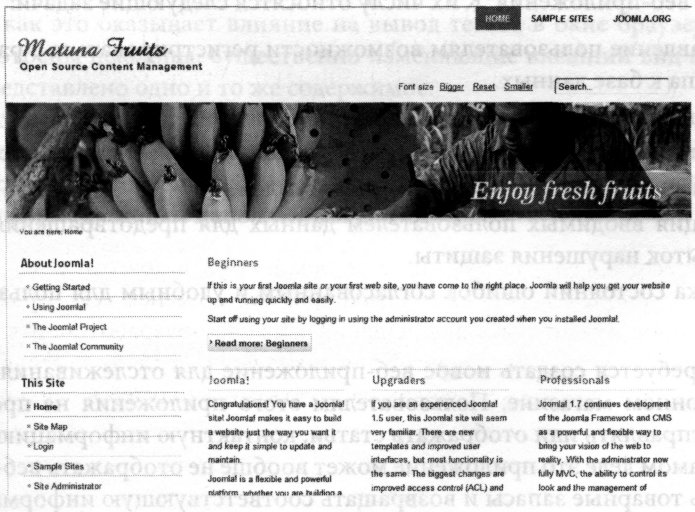


Рис. 1.11. Начальная страница веб-сайта с шаблоном Beez5

Выбор типа расширения для конкретного проекта

Как пояснялось ранее, для расширения функциональных возможностей Joomla имеются четыре варианта выбора: замещения компоновки, подключаемые модули, компоненты и модули. Какой же тип расширения выбрать для конкретного проекта?

Как правило, ответ на этот вопрос становится самоочевидным, как только составится ясное представление о принципе действия каждого типа расширения. Не менее важно понять, что для выполнения некоторых проектов может потребоваться определенное сочетание различных типов расширений. Так, если проект будет в конечном итоге действовать таким же образом, как и базовое расширение, то данное обстоятельство зачастую должно подсказать, какой именно тип расширения следует выбрать для проекта.

Обдумывая свой выбор, можно попытаться решить конкретную проектную задачу самым простым способом. Как правило, к простейшим типам расширения относятся компоновки и подключаемые модули. Следующими по сложности разработки являются модули, а самыми сложными — компоненты. Поэтому если проектную задачу можно решить самым простым способом, то для данного проекта лучше всего выбрать простейшее расширение.

Применение Joomla в качестве платформы или интегрированной среды

Joomla представляет собой систему управления информационным наполнением (контентом) (CMS) веб-сайтов, построенную на основе повторно используемой библиотеки классов, хранящейся, главным образом, в папке `libraries/joomla`. Классы из этой библиотеки выполняют многие низкоуровневые задачи, которые требуются для CMS или любого другого веб-приложения. К их числу относятся следующие задачи:

- Предоставление пользователям возможности регистрироваться с разными правами доступа к базе данных.
- Отслеживание текущего сеанса работы с браузером.
- Буферизация выводимых данных для повышения производительности.
- Обработка событий для подключаемых модулей.
- Фильтрация вводимых пользователем данных для предотвращения злонамеренных попыток нарушения защиты.
- Обработка состояний ошибок согласованным и удобным для пользователей способом.

Допустим, требуется создать новое веб-приложение для отслеживания товарных запасов в электронном магазине. Пользователям этого приложения не предоставляется возможность отправлять или отображать статьи, контактную информацию или крупные заголовки. На самом деле это приложение может вообще не отображать веб-страницы, но лишь проверять товарные запасы и возвращать соответствующую информацию другому приложению.

В данном примере совсем необязательно, да и не нужно устанавливать всю систему Joomla CMS, но может потребоваться проверка полномочий пользователя на выполнение

конкретных действий. Для решения поставленной задачи определенно потребуется взаимодействие с базой данных, фильтрация запросов с целью повысить безопасность самого приложения, а также обработка состояний ошибок. Иными словами, часть Joomla, относящаяся к CMS, не потребуется. Но в то же время можно было бы воспользоваться библиотеками Joomla, чтобы сэкономить немало труда, повторно используя уже написанный код, для чего библиотеки подходят как нельзя лучше.

В версиях 1.5 и 1.6 эту задачу можно было бы решить, установив сначала Joomla, а затем удалив те части, которые не потребуются в нашем приложении. Но это был бы лишний и совсем не обязательный труд. По этой причине разработка функциональных возможностей библиотек Joomla выделена в отдельный проект под названием Joomla Platform Project, начиная с июля 2011 года. Этот проект позволяет тем разработчикам, которым требуется только платформа Joomla без CMS, получить именно то, что им нужно. Он также позволяет выбрать платформу в качестве отдельного от CMS проекта с собственным календарным планом выпуска, кодовой базой и разработчиками. Более подробно платформа Joomla рассматривается в главе 13, где приводится также пример создания приложения с ее помощью.

Резюме

В этой главе было показано, что разработчик, пользующийся Joomla, должен, по меньшей мере, быть знаком с веб-серверами, PHP, SQL, XHTML, XML, CSS и Javascript. Начинающим этого может показаться слишком много, но на самом деле Joomla существенно упрощает веб-разботку.

Кроме того, были рассмотрены различные виды разработок, которые можно выполнять в Joomla: замещения компоновок, подключаемые модули, компоненты и модули. В главе было также показано, каким образом реализуется локализация Joomla на разные языки мира и как это оказывает влияние на вывод текста в окне браузера. Затем были вкратце рассмотрены шаблоны, существенно изменяющие внешний вид веб-сайта, даже если на нем представлено одно и то же содержимое.

После этого были рассмотрены варианты выбора типа расширения для конкретного проекта. И в заключение были рассмотрены вопросы применения Joomla в качестве платформы для построения совершенно отдельных приложений вне сферы действия системы управления информационным наполнением веб-сайтов.

2

Подготовка рабочей станции к разработке в Joomla!

Разработка программ для Joomla! обычно выполняется на локальной рабочей станции. С этой целью следует установить программное обеспечение веб-сервера, необходимое для выполнения Joomla на локальной машине. Кроме того, нужно организовать редактирование исходных файлов PHP и прочих файлов, составляющих приложение Joomla.

Правда, все это программное обеспечение свободно доступно для загрузки и может быть установлено во всех распространенных операционных системах, включая Windows, Mac OS X и Linux. Кроме того, имеется немало других свободно доступных средств для правки программных файлов Joomla, как будет показано далее в этой главе, где поясняется также порядок настройки рабочей станции для выполнения Joomla, создания и правки файлов, необходимых для написания прикладных программ.

Если вы уже настроили свою рабочую станцию под Joomla и разработку средствами Joomla, можете пропустить раздел "Требования к выполнению Joomla". А если вы уже пользуетесь такой интегрированной средой разработки, как Eclipse или NetBeans, то можете не читать и раздел "Рабочие инструменты".

Требования к запуску Joomla

Как пояснялось в главе 1, для нормальной работы Joomla требуется веб-сервер (Apache или Microsoft Internet Information Services — IIS), язык программирования PHP и база данных (чаще всего MySQL). Каждый из этих программных компонентов может быть загружен и установлен отдельно от соответствующих веб-сайтов. Но, как правило, Apache, PHP и MySQL проще установить общим пакетом. А в Mac OS X и Linux эти программные компоненты могут быть уже установлены по умолчанию.

Следует иметь в виду, что независимо от способа установки (по отдельности или общим пакетом) конечный результат остается неизменным: Apache, PHP и MySQL устанавливаются на локальной машине. Единственное отличие состоит в том, что каждый пакет может немного отличаться своей комплектацией, именами папок и отдельными параметрами настройки по умолчанию.



Важное примечание по поводу безопасности

В этой главе предполагается, что для разработки и тестирования создается локальная среда, а тестируемые веб-сайты не связаны со Всемирной паутиной (или просто веб). Следовательно, вопросы безопасности веб-сервера не столь актуальны, как в том случае, когда веб-приложение размещается на реальном сервере. Это, разумеется, не означает, что не нужно уделять внимание защите самих программ, разрабатываемых для Joomla, от внешних атак.

Если вы работаете на удаленном сайте, доступном в веб, непременно организуйте его защиту — даже на время разработки. Это можно сделать самыми разными способами, в том числе используя файл с расширением `.htaccess` в Apache, требующий ввода пароля для доступа по указанному URL. В таком случае вам, возможно, потребуется консультация специалиста по безопасности веб вообще и Joomla в частности.

Папка `DocumentRoot` в Apache

Обо всех упомянутых выше программных компонентах и средах нужно знать следующее. Как пояснялось ранее, веб-сервер Apache представляет собой программное обеспечение связи браузера с веб-сайтом. Если веб-сервер Apache устанавливается локально, то URL по веб-адресу `http://localhost` сопоставляется с папкой на локальной рабочей станции. Как только этот URL будет введен в браузере, Apache произведет поиск файла `index.php` или `index.html` в папке `DocumentRoot`, находящейся на локальном компьютере. Если установлен пакет веб-сервера XAMPP, то папка `DocumentRoot` обычно находится по пути `C:\xampp\htdocs`, а если установлен пакет WampServer — то по пути `C:\Program Files\wamp\www`.

Папку `DocumentRoot` можно заменить на другую, внеся соответствующие коррективы в файл `httpd.conf`, находящийся в папке `apache/conf`. Например, в пакете XAMPP этот файл находится по пути `c:\xampp\apache\conf`. А в пакете WampServer файл `httpd.conf` можно отредактировать в самом приложении, выбрав команду Config Files (Конфигурационные файлы)⇨`httpd.conf`.

Определившись с папкой `DocumentRoot`, вы можете затем выбрать место для установки файлов Joomla. Поиск подходящего места начнется с любого URL, введенного в браузере с префиксом `http://localhost`, т.е. с папки `DocumentRoot`, и следуя далее вниз по древовидной структуре папок. Так, если корневая папка для документов находится по пути `c:\xampp\htdocs`, то Joomla можно установить в папке `c:\xampp\htdocs\joomla_development\joomla_25`. В этом случае URL, который нужно ввести для доступа к административной части данной установки, будет следующим: `http://localhost/joomla_development/joomla_25/administrator`. Веб-сервер Apache будет искать файл `index.php` в папке `joomla_development/joomla_25/administrator`, вложенной в папку `DocumentRoot`.

Конечно, Joomla можно было бы установить и в самой папке `DocumentRoot`, но тогда в ней нельзя было бы установить и все остальное. А следуя приведенному выше примеру установки, можно одновременно начать несколько проектов в отдельных подпапках, вложенных в папку `DocumentRoot`. Итак, с учетом всего сказанного выше, перейдем к рассмотрению различных вариантов установки требующегося программного обеспечения на разных платформах.

Получение своевременных инструкций

Программы и комплекты, рассматриваемые в этом разделе, постоянно обновляются новыми версиями. Со временем выходят также новые версии Windows, Linux и Mac OS X. Поэтому материал этого раздела актуален лишь на момент написания книги. А впоследствии он может устареть или оказаться неполными в силу неизбежных изменений в программном обеспечении как самого веб-сервера, так и платформы, для которой он предназначен.

По этой причине для каждого комплекта здесь приводятся ссылки, по которым нетрудно получить своевременные инструкции. Помимо этих ссылок, здесь приводится также ссылка на статью, опубликованную на веб-сайте документации по Joomla, с подробными инструкциями по установке (`http://docs.joomla.org/Setting_up_your_workstation_for_Joomla!_development`). Эта статья обновляется по мере изменений в соответствующем программном обеспечении.

Платформа Windows

В различных версиях Windows (Windows 7, Windows Vista и Windows XP) программные компоненты Apache, PHP и MySQL не устанавливаются по умолчанию. Двумя наиболее распространенными стандартными пакетами для установки динамической веб-среды в Windows являются XAMPP и WAMPP.

Наиболее распространенным в Windows считается пакет XAMPP (`http://www.apachefriends.org/en/xampp.html`). Он легко устанавливается и хорошо документирован. Кроме того, он доступен в Linux и Mac OS X. Подробные инструкции по его установке в Windows приведены по адресу `http://www.apachefriends.org/en/xampp-windows.html`.

Еще одним стандартным для Windows является пакет WampServer. К его преимуществам относится простота перехода к другим версиям Apache, PHP и MySQL. И это весьма значительное преимущество, если, например, требуется протестировать программу в разных версиях PHP. Пакет WampServer доступен по адресу `http://www.wampserver.com/en` или `http://www.wampserver.com/ru`. Он очень просто устанавливается в Windows и хорошо документирован на сайте по указанному выше адресу.

Примечание по поводу Microsoft IIS и SQL Server

Joomla может выполняться как на веб-сервере Apache, так и на веб-сервере Microsoft IIS. Но поскольку веб-сервер Apache имеет более долгую историю поддержки, а большинство веб-сайтов Joomla работают именно на нем, то разработчики в основном отдают предпочтение именно ему. По этим причинам в качестве сервера для разработки веб-сайтов рекомендуется Apache.

С версии 2.5 в Joomla поддерживается также база данных Microsoft SQL Server. Но в качестве основной базы данных для разработки веб-приложений в Joomla рекомендуется все же MySQL.

Платформа Mac OS X

Как упоминалось выше, версия пакета XAMPP имеется и для платформы Mac OS X. Ее можно загрузить по адресу `http://www.apachefriends.org/en/xampp-macosx.html`. В этом случае папка `DocumentRoot` по умолчанию находится по пути `/Applications/XAMPP/htdocs/`.

Имеется также пакет MAMP, предназначенный исключительно для платформы Mac OS X. Его можно загрузить по адресу `http://www.mamp.info/en/mamp/index.html`. В этом случае папка `DocumentRoot` по умолчанию находится по пути `/Applications/MAMP /htdocs/`.

В Mac OS X и Linux имеется также возможность воспользоваться версиями Apache, MySQL и PHP, распространяемыми как составная часть собственной операционной системы. А поскольку Mac OS X является производной от Linux, то эти программные компоненты могут работать и в Mac OS X как ее собственные.

Если у вас нет опыта работы с этими программными компонентами, вам, вероятно, будет проще выбрать один из существующих стандартных пакетов, чем пользоваться собственными их версиями. Кроме того, стандартные пакеты XAMPP и MAMP, как правило, содержат более актуальные версии Apache, MySQL и PHP, чем версии, входящие в комплект с операционной системой Mac OS X.

Важное примечание

Веб-сервер Apache может быть установлен по умолчанию в операционной системе Mac OS X. В этом случае он определяет порт на компьютере как "прослушивающий". Если же затем установить вторую версию Apache из пакета XAMPP или MAMP, то и он попытается воспользоваться тем же самым портом, а следовательно, не сможет работать корректно.

Во избежание подобного усложнения придется изменить один из портов. Проще всего, вероятно, изменить порт в пакете XAMPP или MAMP на что-нибудь вроде **8080**. В пакете XAMPP для этого достаточно отредактировать файл `\Applications\XAMPP\xamppfiles\etc\httpd.conf`, изменив в нем строку `Listen 80` на `Listen 8080`. А в пакете MAMP порт можно изменить на панели глобальных параметров настройки, что, в свою очередь, внесет коррективы в файл `httpd.conf`. И в этом случае прослушивающий порт следует изменить на `Listen 8080`.

Платформа Linux

Все, что было сказано выше относительно платформы Mac OS X, справедливо в основном и для платформы Linux. И в этом случае имеется возможность выбрать пакет XAMPP или собственные версии Apache, MySQL и PHP. Начинающим проще установить пакет XAMPP. Сведения о загрузке пакета XAMPP для платформы Linux находятся по адресу `http://www.apachefriends.org/en/xampp-linux.html`.

По умолчанию папка `DocumentRoot` находится по пути `/opt/lampp/htdocs/`. Как пояснялось выше для платформы Mac OS X, вам, возможно, придется изменить используемый по умолчанию прослушивающий порт, если в вашей системе уже работает экземпляр веб-сервера Apache. В пакете XAMPP для этой цели придется внести соответствующие коррективы в файл `/opt/lampp/etc/httpd.conf`. А состоят эти коррективы в замене строки `Listen 80` строкой `Listen 8080`.

Владелец файлов и папок по умолчанию

Права доступа к файлам и папкам в Linux и Mac OS X по умолчанию могут привести к сложностям при попытке установить и отредактировать файлы Joomla. Поэтому следует убедиться в том, что соответствующему пользователю Linux и Mac OS X разрешено выполнять запись в ту папку, где устанавливается Joomla и размещаются файлы Joomla.

Владелец всех файлов, созданных веб-сервером Apache, устанавливается с помощью параметров `User` и `Group` в файле `httpd.conf`. По умолчанию оба эти параметра настроены на "демона" (службу, работающую в фоновом режиме). А это означает, что у пользователя нет полномочий на запись в файлы и папки Joomla. Если речь идет о локальном сайте, предназначенном в основном для целей разработки и тестирования, то параметры `User` и `Group` можно настроить на зарегистрированного в системе пользователя. Благодаря этому будут получены полномочия на создание и удаление файлов и папок Joomla.

Рабочие инструменты

Разработчики Joomla могут пользоваться самыми разными инструментами для создания и редактирования программных файлов Joomla. Так, если задать в дискуссионной группе разработчиков программного обеспечения следующий вопрос: какие инструментальные средства они предпочитают для написания программ, то ответов на него, скорее всего, будет столько же, сколько и мнений по поводу достоинств и недостатков этих самых инструментальных средств. Аналогичное разнообразие мнений сложилось и относительно инструментальных средств, применяемых разработчиками в Joomla. Не существует единого, самого лучшего инструмента для всех и на все случаи жизни. Поэтому в последующих разделах мы постараемся как можно более объективно проанализировать достоинства и недостатки различных инструментальных средства для разработки в Joomla.

Интегрированная среда разработки в сравнении с текстовым редактором

При установке Joomla весь исходный код программы загружается в систему в виде простых текстовых файлов. К числу основных типов файлов в базовой установке относятся PHP (с расширением `.php`), XML (с расширением `.xml`), JavaScript (с расширением `.js`), CSS (с расширением `.css`), а также SQL (с расширением `.sql`). Все подобные типы определяют простые текстовые файлы, а это означает, что они могут быть отредактированы в любом текстовом редакторе. Следовательно, у вас имеется возможность поправить уже имеющиеся файлы Joomla и создать новые без специального программного обеспечения.

В таком случае зачем обременять себя установкой и освоением нового программного обеспечения только для целей разработки в Joomla? Ответ на этот вопрос очевиден: производительность труда — по крайней мере, для многих разработчиков.

Назначение интегрированной среды разработки (IDE) в том и состоит, чтобы повысить производительность труда разработчиков. Такие интегрированные среды разработки, как, например, Eclipse или NetBeans, подробнее рассматриваемые далее в этой главе, предоставляют следующие функциональные возможности для упрощения процесса программирования на PHP.

- Обнаружение синтаксических ошибок в коде по мере его набора.
- Автоматическое завершение набранных частей кода на основании известного синтаксиса.
- Автоматическое отображение сведений о методах и полях редактируемого класса.
- Сравнение двух версий исходного файла (или двух разных исходных файлов) для выявления в них конкретных отличий.
- Упрощение поиска и замены во всей кодовой базе.
- Отладка программ на PHP.

Для выполнения всех перечисленных выше операций совсем не обязательно иметь IDE. Но все эти операции можно выполнять в IDE, как в едином инструментальном средстве.

Еще одно преимущество IDE состоит в том, что она сводит различные задачи разработки в единый проект. В частности, многие разработчики Joomla пользуются системой контроля версий типа Git или Subversion. К этим системам подключаются модули, работающие с IDE, что дает возможность пользоваться системой контроля версий непосредственно в IDE. Другим примером служит программа автоматического построения проектов типа Phing или Ant. И эти программы можно выполнять непосредственно в IDE, что намного упрощает труд многим разработчикам, особенно в крупных проектах.

Но некоторым применение IDE дается дорогой ценой, поскольку освоение самой среды требует дополнительного времени и труда. Кроме того, программные ошибки могут встречаться в IDE чаще, чем в текстовом редакторе, в силу ее большей сложности. И поэтому одни считают, что возникающая в связи с этим неудовлетворенность не стоит потенциального выигрыша в производительности труда. А другие легко мирятся с перспективой освоения нового программного обеспечения и возможными его недоработками, поскольку считают эти недостатки малой ценой, которую приходится платить за несомненные преимущества IDE.

Если вы собираетесь много заниматься разработкой веб-сайтов в Joomla, вам, вероятно, стоит уделить время хотя бы опробованию IDE, чтобы выяснить, насколько эта среда вам подходит. А если вы не собираетесь тратить время и силы на освоение IDE, но желаете сразу же приступить к написанию кода, то IDE вряд ли вам подойдет.

Варианты IDE с открытым кодом

Двумя самыми распространенными IDE для Joomla являются Eclipse и NetBeans. Обе IDE отлично справляются со своей задачей, помогая разработчику работать более продуктивно. В последующих подразделах они рассматриваются по очереди.

Eclipse

Несмотря на то что Eclipse давно уже стала самой распространенной IDE для разработки приложений на Java, в программировании на PHP она стала применяться относительно недавно. В 2007 году специально для разработчиков на PHP были выпущены инструментальные средства, которые так и называются: PHP Development Tools (PDT). Они представляют собой набор подключаемых модулей, адаптирующих Eclipse для работы с кодом PHP.

На момент написания этой книги самой последней считалась версия PDT 3.0. В рамках проекта Eclipse новая версия выпускается каждый год в июне, а вместе с ней зачастую и новая версия PDT. Пакет Eclipse можно свободно загрузить по адресу `http://eclipse.org/downloads`. Для разработки в Joomla обычно выбирается вариант `Eclipse for PHP Developers` (Eclipse для разработчиков программ на PHP) этого пакета. Подробные инструкции по установке и настройке Eclipse для разработки в Joomla можно найти на сайте документации по Joomla, перейдя по адресу `http://docs.joomla.org/Setting_up_your_workstation_for_Joomla!_development`.

IDE Eclipse написана на языке Java и работает на платформах Windows, Linux и Mac OS X. Это свободно доступное программное обеспечение с открытым кодом. И хотя не существует официальной статистики по поводу того, каким именно IDE отдают предпочтение разработчики Joomla, тем не менее Eclipse считается едва ли не самой распространенной IDE среди опытных разработчиков данной категории, и не в последнюю очередь потому, что она оказалась первой свободно доступной IDE для разработки на PHP.

Рассмотрим вкратце, чем же Eclipse может помочь в разработке веб-сайтов средствами Joomla. Прежде всего следует обратить внимание на ключевое понятие проекта для таких IDE, как Eclipse или NetBeans. В большинстве текстовых редакторов исходный файл правится постепенно, и самому редактору вообще ничего не известно о других файлах, связанных с исходным, да это ему и не нужно. А в IDE сначала создается проект, в котором хранится группа связанных вместе файлов. Так, если вы работаете над проектом Joomla, то все приложение Joomla содержится в одном проекте. Поначалу такой подход может показаться не совсем понятным, но он вполне оправдан с точки зрения разработки, поскольку разработчикам все время приходится работать с целыми группами связанных вместе файлов.

Создавая проект PHP в Eclipse, вы получаете представление под названием `PHP Explorer` (Проводник по PHP), где можете просматривать весь проект и открывать любой файл для редактирования. В представлении `PHP Explorer`, приведенном в качестве примера на рис. 2.1, один файл открыт для редактирования в левом верхнем углу.

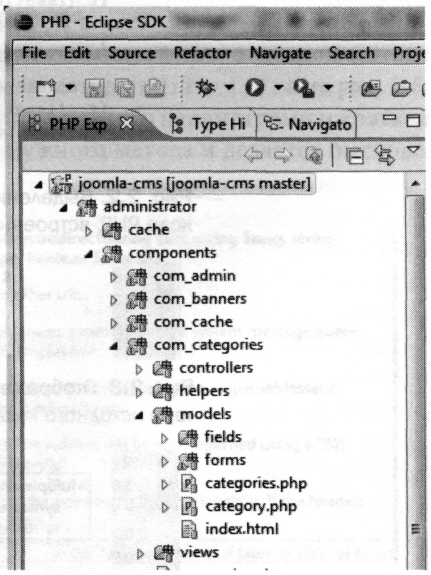


Рис. 2.1. Пример представления `PHP Explorer` в Eclipse

А теперь рассмотрим в качестве примера некоторые из упоминавшихся ранее функциональных возможностей IDE. Начнем с выделения синтаксиса. На рис. 2.2 показана программа на PHP, открытая для редактирования в Eclipse. Различные

синтаксические конструкции выделены разным цветом для удобства чтения ее кода. Посмотрим далее, что произойдет, если сделать ошибку в коде. Как показано на рис. 2.3, после оператора `if` оставлены скобки.

Как видите, слева от строки с синтаксической ошибкой появляется знак **X**, а в самой строке обнаруженные ошибки подчеркнуты волнистой линией (на экране эти признаки ошибок отображаются красным цветом). Если навести курсор на знак **X**, то появится всплывающая текстовая подсказка, как показано на рис. 2.4. Эта подсказка вкратце поясняет характер ошибки и указывает возможный способ ее устранения.

Еще одним важным свойством IDE является автоматическое завершение кода. Оно представлено в разных формах. В простейшей форме символы предоставляются попарно. Например, при наборе исходного кода PHP вместе с левой круглой скобкой практически всегда требуется и правая скобка, поэтому редактор автоматически предоставляет обе скобки. Это же относится и к фигурным скобкам, кавычкам и элементам разметки в коде HTML.

```
/**
 * Class constructor.
 *
 * @param   array  $config  A configuration arra
 *                          session_name, client
 *
 * @since   11.1
 */
public function __construct($config = array())
{
    jimport('joomla.utilities.utility');
    jimport('joomla.error.profiler');

    // Set the view name.
    $this->_name = $this->getName();

    // Only set the clientId if available.
    if (isset($config['clientId'])) {
        $this->_clientId = $config['clientId'];
    }

    // Enable sessions by default.
    if (!isset($config['session'])) {
        $config['session'] = true;
    }
```

Рис. 2.2. Выделение синтаксиса в редакторе исходного кода PHP, встроенном в Eclipse

```
 98          $if $myVar != 1 {
 99              echo 'something is wrong';
100          }
101
```

Рис. 2.3. Отображение синтаксической ошибки в редакторе исходного кода PHP, встроенном в Eclipse

```
 97
 98
 99                                    is wrong';
100
101
```

Multiple annotations found at this line:
- syntax error, unexpected '{'
- syntax error, unexpected '$myVar'

Рис. 2.4. Всплывающая текстовая подсказка, раскрывающая характер ошибки

Еще одна форма завершения кода имеет место в том случае, если требуется набрать имя метода. В качестве примера на рис. 2.5 набрано выражение `$x = $this->` и затем нажата комбинация клавиш <Ctrl+Пробел>.

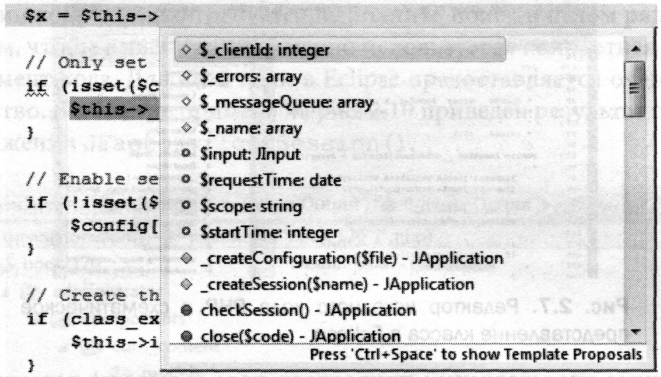


Рис. 2.5. Выбор подходящего метода в редакторе исходного кода PHP, встроенном в Eclipse

Редактору известно, что `$this` — это объект типа `Japplication`, и поэтому он отображает все поля и методы, доступные в данном классе. Для того чтобы выбрать подходящий метод или поле, достаточно щелкнуть кнопкой мыши на его имени или воспользоваться клавишами с изображением стрелок. Как только поле или метод будут выбраны, их имя автоматически вставится в исходный текст программы. Благодаря этому ускоряется набор исходного текста программы и уменьшается вероятность совершения опечаток. По мере набора исходного текста окно поиска автоматически обновляется, чтобы отображать только те имена, которые совпадают с набранным текстом.

Кроме того, в этом окне предоставляются дополнительные сведения о каждом методе или поле. В качестве примера на рис. 2.6, *слева*, выделен метод `redirect()`, а на рис. 2.6, *справа*, представлена форма объявления этого метода со всеми его параметрами и краткая документация на него. Это очень удобно для поиска нужного метода и делается быстрее, чем в отдельном файле или окне.

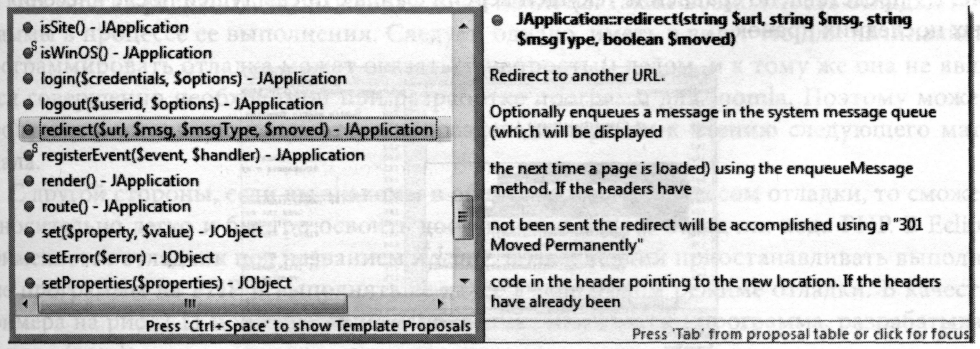


Рис. 2.6. Список параметров и документация на выбранный метод, предоставляемые в редакторе исходного кода PHP, встроенном в Eclipse

Когда правится исходный файл, в Eclipse предоставляется список полей и методов для данного файла в отдельном представлении. Так, на рис. 2.7, *слева*, показана область правки, а на рис. 2.7, *справа*, — соответствующее схематическое представление (`Outline`).

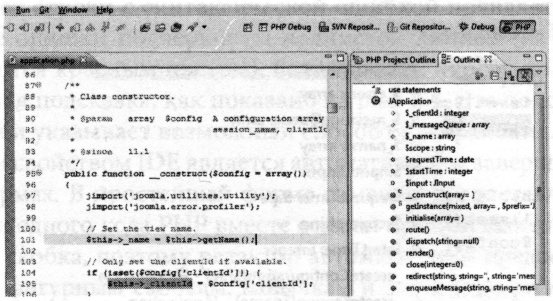


Рис. 2.7. Редактор исходного кода PHP и схематическое представление класса в Eclipse

В схематическом представлении приводятся поля и методы редактируемого класса. А в схематическом представлении проекта PHP (`PHP Project Outline`) можно посмотреть иерархическую структуру всего проекта, как показано на рис. 2.8. В этом представлении можно найти любой класс, доступный в Joomla, и посмотреть всего его поля и методы.

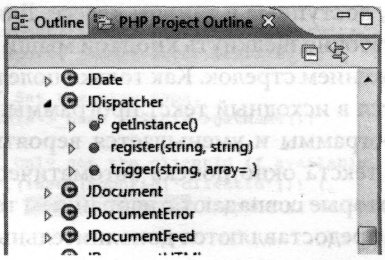


Рис. 2.8. Схематическое представление проекта PHP в Eclipse

Еще одним удобным свойством такой IDE, как Eclipse, является возможность быстро сравнивать две версии исходного файла, чтобы выявить конкретные изменения. Так, на рис. 2.9 представлено сравнение текущей версии файла с предыдущей после внесения самых последних правок.

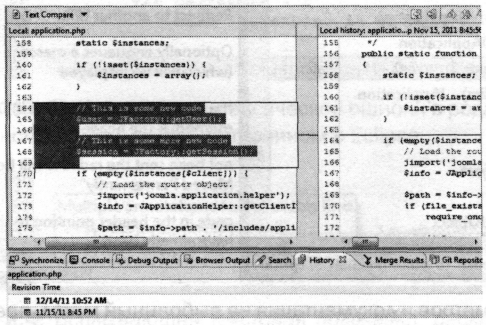


Рис. 2.9. Пример сравнения двух версий исходного файла в Eclipse

Этим свойством можно воспользоваться для сравнения различных версий одного и того же исходного файла. Их можно извлечь из местной предыстории, автоматически сохраняемой в Eclipse, или же из хранилища версии, если применяется система контроля версий.

При написании кода нередко требуется выполнить поиск в целом ряде программ или в проекте в целом, чтобы выяснить, где именно используется конкретная переменная или отдельный фрагмент кода. Для этой цели в Eclipse предоставляется очень простое и эффективное средство. В качестве примера на рис. 2.10 приведен результат поиска в проекте вхождений выражения `JFactory::getSession()`.

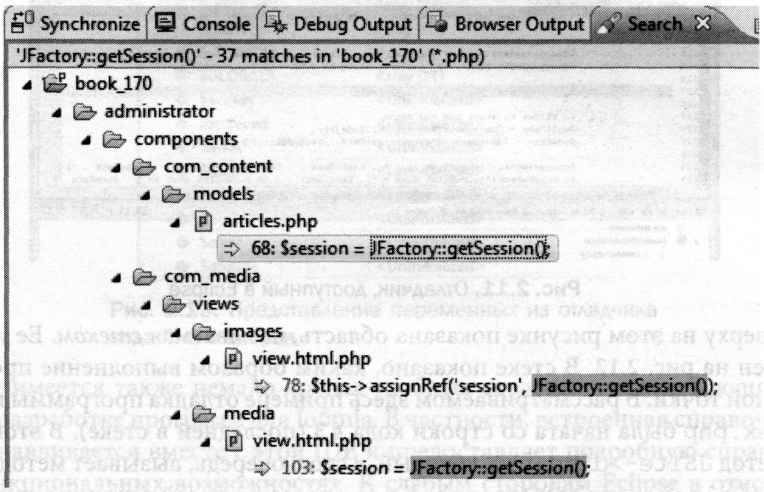


Рис. 2.10. Пример поиска выражения в проекте средствами Eclipse

Ради удобства результаты поиска привязываются к редактору исходного текста. Так, если дважды щелкнуть на одной из строк кода в результатах поиска, то в редакторе исходного текста откроется целевой файл, а курсор установится на номере той строки, где был найден искомый текст. В этом представлении можно выполнять не только поиск, но и замену найденного текста.

Для многих разработчиков очень важной является также возможность отладки программы в процессе ее выполнения. Следует, однако, иметь в виду, что для начинающих программировать отладка может оказаться непростым делом, и к тому же она не является совершенно необходимой при разработке программ для Joomla. Поэтому можете пропустить остальную часть этого подраздела и перейти к чтению следующего материала.

С другой стороны, если вы знакомы в общих чертах с процессом отладки, то сможете относительно легко и быстро освоить доступный в Eclipse отладчик кода PHP. В Eclipse применяется отладчик под названием *Xdebug*, позволяющий приостанавливать выполнение программы на PHP и выполнять ее далее в пошаговом режиме отладки. В качестве примера на рис. 2.11 показана приостановленная для отладки программа, разрабатываемая для Joomla.

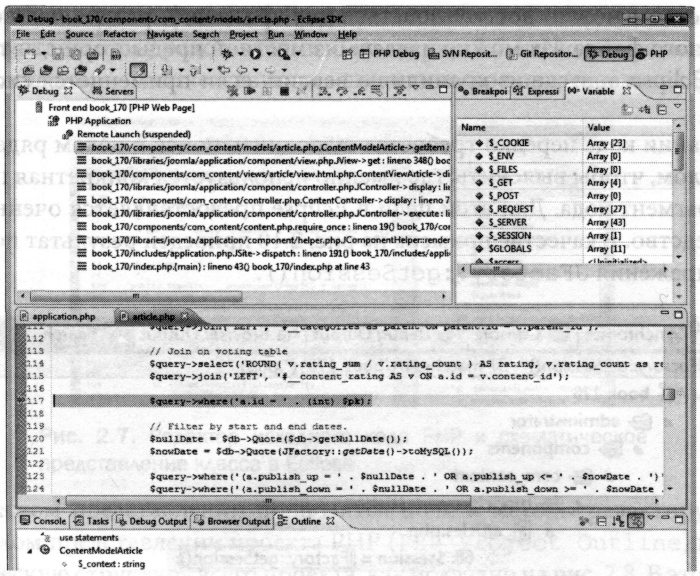


Рис. 2.11. Отладчик, доступный в Eclipse

Слева вверху на этом рисунке показана область, называемая *стеком*. Ее увеличенный вид приведен на рис. 2.12. В стеке показано, каким образом выполнение программы дошло до данной точки. В рассматриваемом здесь примере отладка программы из исходного файла `index.php` была начата со строки кода **43** (последней в стеке). В этой строке вызывается метод `JSite->dispatch`, который, в свою очередь, вызывает метод компонента `JComponentHelper::renderComponent` и так далее до самой вершины стека, где вызывается метод `ContentModelArticle->getItem`.

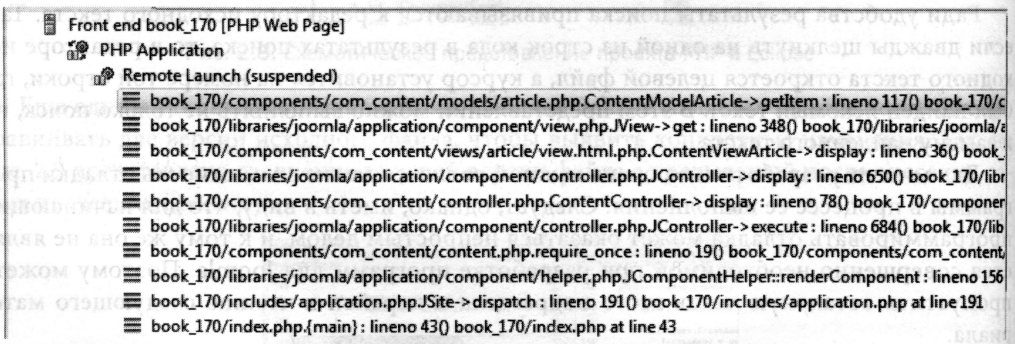


Рис. 2.12. Пример стека из отладчика кода PHP в Eclipse

Важное примечание

Если вы не знаете, что такое стек, — не отчаивайтесь. Принцип его организации и действия подробно рассматривается в главе 3.

В представлении переменных (`Variables`), приведенном на рис. 2.13, отображаются значения всех переменных из области действия в данной точке выполнения программы.

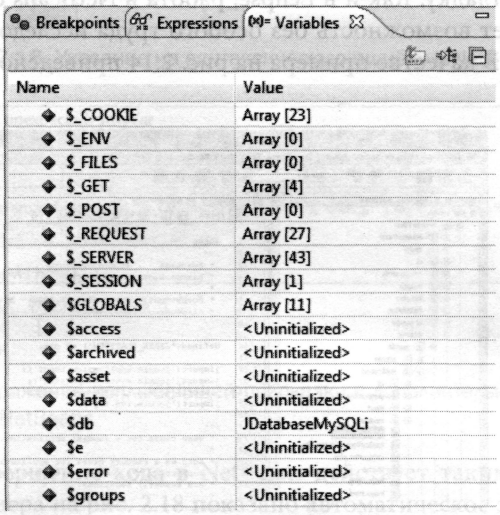


Рис. 2.13. Представление переменных из отладчика кода PHP в Eclipse

В Eclipse имеется также немало других удобных средств, способствующих более эффективной разработке программ для Joomla. В частности, встроенная справочная система Eclipse устанавливается вместе с этой IDE и предоставляет подробную справку о ее различных функциональных возможностях. К слабым сторонам Eclipse в отношении программирования на PHP можно отнести недостаточное количество учебного материала и пользовательской документации, хотя на сайте документации по Joomla опубликован ряд полезных статей, посвященных Eclipse.

PHPEclipse

PHPEclipse (`http://www.phpeclipse.com`) — это еще одна IDE для программирования на PHP с функциональными возможностями, аналогичными программе PDT для Eclipse. Она поддерживается вне рамок официального проекта семейства Eclipse. На момент написания этой книги активная разработка PHPEclipse, по-видимому, была приостановлена, хотя эта IDE по-прежнему доступна для загрузки.

NetBeans

Другой IDE с открытым кодом, считающейся основной для разработки программ на PHP, является NetBeans. Она свободно доступна для загрузки по адресу `http://netbeans.org/downloads/index.html`. Аналогично Eclipse, IDE NetBeans была первоначально разработана для программирования на Java и теперь поддерживает разработку приложений на других языках программирования.

К главным преимуществам NetBeans относится отличная документация для разработки программ на PHP, включая и учебный видеоматериал. Именно по этой причине NetBeans является отличным выбором для тех, кто только начинает осваивать IDE.

NetBeans обладает такими же функциональными возможностями, как и Eclipse, включая выделение синтаксиса, автоматическое завершение кода, сравнение исходных файлов, глобальный поиск и отладку. Как и в Eclipse, работа в NetBeans организуется по отдельным проектам, что дает возможность без особого труда исследовать проект и находить в нем нужные файлы. В качестве примера на рис. 2.14 приведено представление проекта (`Project`) в NetBeans.

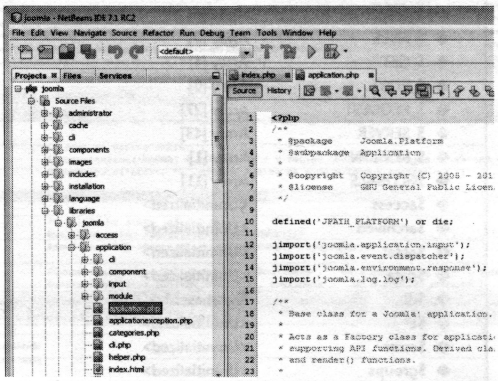


Рис. 2.14. Пример представления проекта в NetBeans

Для разработки программ на PHP в NetBeans поддерживается выделение синтаксиса, выдача сообщений об ошибках и автоматическое завершение кода. В качестве примера на рис. 2.15 показано, каким образом в NetBeans осуществляется выделение синтаксиса, а на рис. 2.16 — сообщается об ошибке в коде PHP. Если навести курсор на маркер ошибки, появятся дополнительные сведения об ошибке, как показано на рис. 2.17.

```
/**
 * Class constructor.
 *
 * @param    array  $config  A configuration array
 *                           session_name, clientId
 *
 * @since    11.1
 */
public function __construct($config = array())
{
    jimport('joomla.utilities.utility');
    jimport('joomla.error.profiler');

    // Set the view name.
    $this->_name = $this->getName();

    // Only set the clientId if available.
    if (isset($config['clientId'])) {
        $this->_clientId = $config['clientId'];
    }

    // Enable sessions by default.
    if (!isset($config['session'])) {
        $config['session'] = true;
    }
```

Рис. 2.15. Выделение синтаксиса в NetBeans

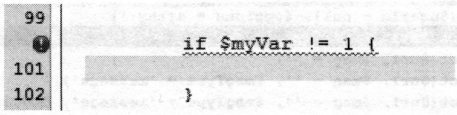


Рис. 2.16. Указание на синтаксическую ошибку в NetBeans

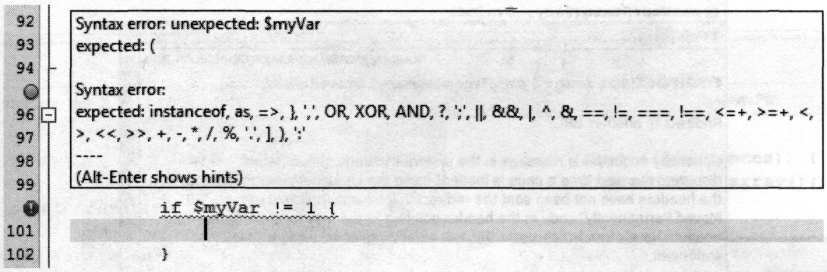


Рис. 2.17. Отображение всплывающей подсказки с дополнительными сведениями об ошибке в NetBeans

Автоматическое завершение кода в NetBeans действует таким же образом, как и в Eclipse. В качестве примера на рис. 2.18 показано автоматическое завершение вызова метода из класса `Japplication` после набора выражения `$x = $this->` и нажатия комбинации клавиш <Ctrl+Пробел>. Если набрать первую часть вызова нужного метода, то, как и в Eclipse, список методов, соответствующих набранному тексту, сужается. Если же выделить метод в исходном тексте программы, появится блок документации на этот метод, как показано на рис. 2.19.

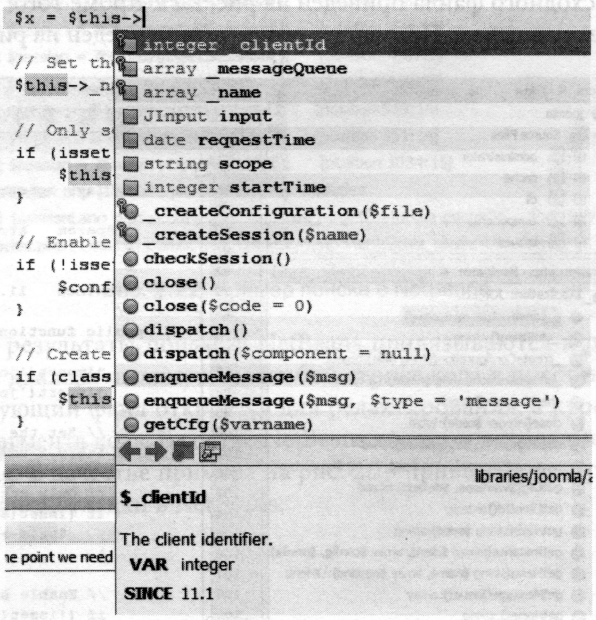


Рис. 2.18. Автоматическое завершение кода в NetBeans

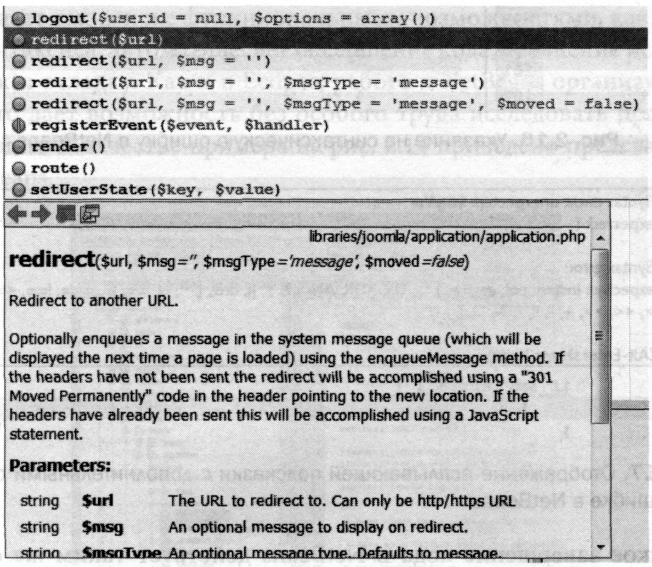


Рис. 2.19. Краткая документация на метод, появляющаяся при автоматическом завершении кода в NetBeans

В схематическом представлении редактируемого класса в NetBeans отображаются методы и поля этого класса. Соответствующий пример приведен на рис. 2.20. В NetBeans поддерживается также сравнение текущей и предыдущей версий исходного файла с использованием предыстории (локальной или контроля версий). Пример сравнения с предыдущей версией исходного файла приведен на рис. 2.21. Кроме того, в NetBeans допускается поиск по проекту в целом. Пример такого поиска приведен на рис. 2.22.

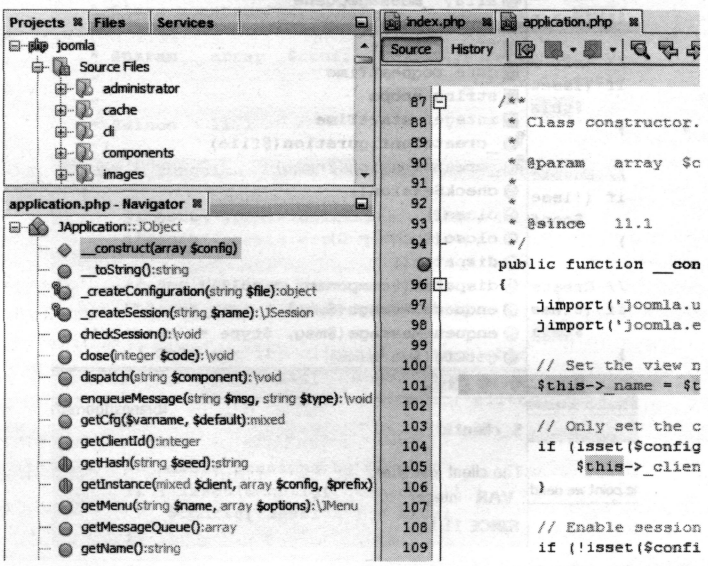


Рис. 2.20. Навигатор NetBeans, отображающий методы редактируемого класса

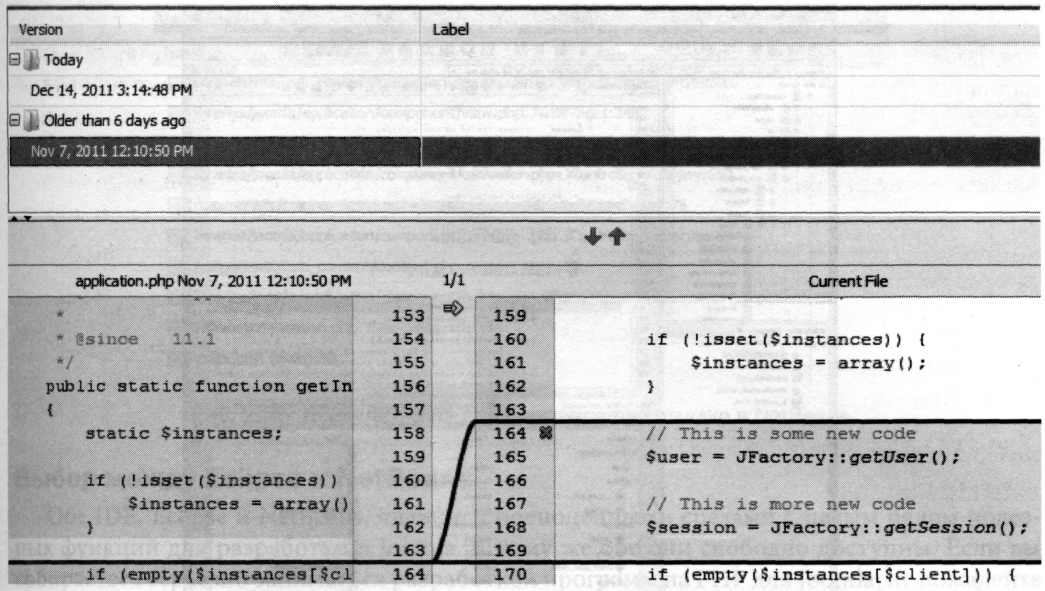


Рис. 2.21. Редактор сравнения версий исходного файла в NetBeans

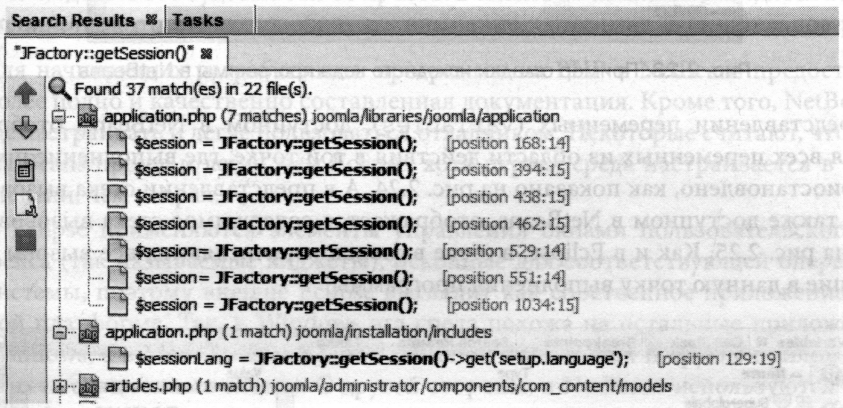


Рис. 2.22. Пример поиска в NetBeans

Как и в Eclipse, результаты поиска в NetBeans привязываются к редактору исходного текста. Так, если дважды щелкнуть на любой строке кода в преставлении результатов поиска, соответствующий файл откроется для редактирования, а курсор установится на месте искомого фрагмента кода. В состав NetBeans входит и встроенный отладчик, также называемый Xdebug. В качестве примера на рис. 2.23 приведена программа, приостановленная во время сеанса отладки в NetBeans.

3.3ak 3469

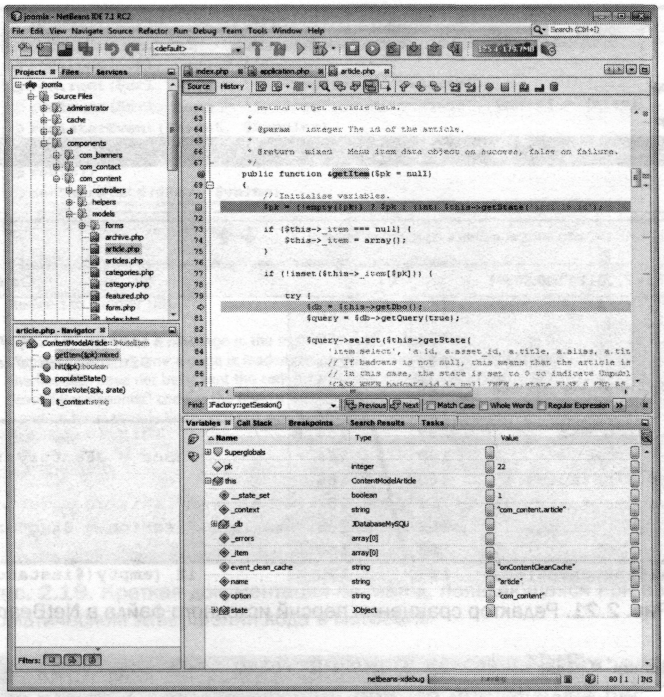


Рис. 2.23. Пример отладки исходного кода программы в NetBeans

В представлении переменных (`Variables`), доступном в NetBeans, отображаются значения всех переменных из области действия в той точке, где выполнение программы было приостановлено, как показано на рис. 2.24. А в представлении стека вызовов (`Call Stack`), также доступном в NetBeans, отображается содержимое стека вызовов, как показано на рис. 2.25. Как и в Eclipse, в стеке вызовов перечисляются все вызовы методов, приведшие в данную точку выполнения программы.

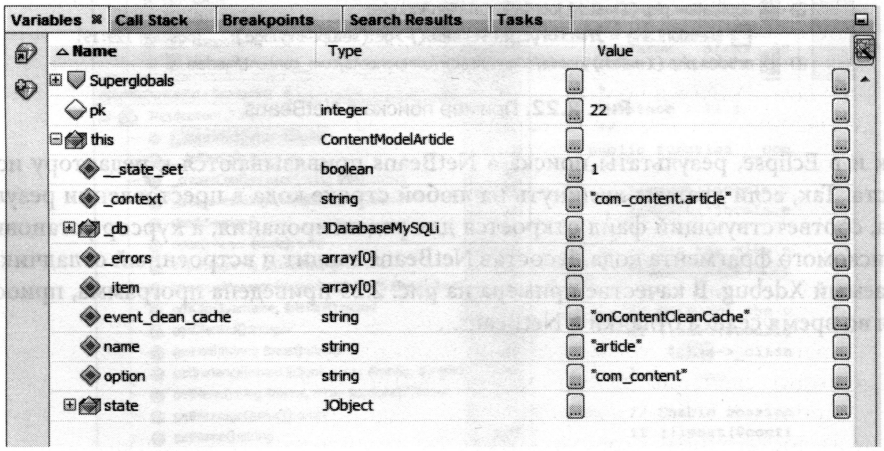


Рис. 2.24. Представление переменных при отладке в NetBeans

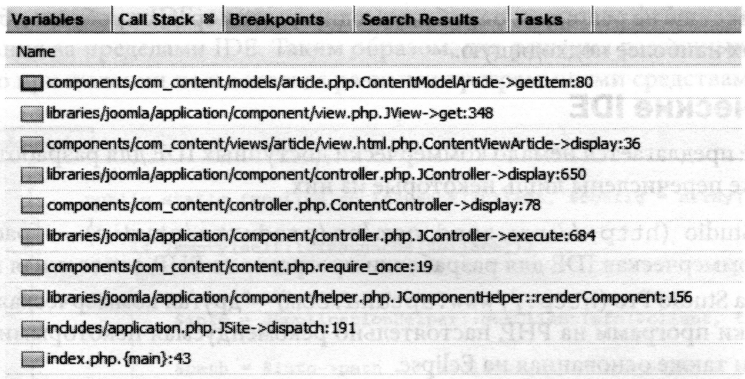


Рис. 2.25. Представление стека вызовов при отладке в NetBeans

Выбор между Eclipse и NetBeans

Обе IDE, Eclipse и NetBeans, являются полноценными средами с целым рядом полезных функций для разработки в Joomla. К тому же обе они свободно доступны. Если вы собираетесь серьезно заниматься разработкой программ на PHP для Joomla, не пожалейте времени на опробование этих IDE, чтобы выбрать из них наиболее подходящую.

Впрочем, сделать это будет на так-то просто. В связи с этим ниже приводятся некоторые соображения, на которые стоит обратить внимание, принимая окончательное решение.

- Для начинающих разрабатывать программы на PHP в NetBeans предоставляется более полно и качественно составленная документация. Кроме того, NetBeans проще настраивать и легче пользоваться отладчиком. Некоторые считают, что освоить NetBeans проще и легче, чем Eclipse, хотя первая среда настраивается в меньшей степени, чем вторая.
- В Eclipse применяются элементы управления окнами пользовательского интерфейса (так называемые виджеты), исконные для соответствующей операционной системы, поэтому внешне Eclipse выглядит как собственное приложение на каждой платформе. Так, в Windows эта среда похожа на остальные приложения для Windows, а в Mac OS X или Linux выглядит так же, как и другие приложения для этих операционных систем. С другой стороны, в NetBeans используются виджеты типа Java SWING, и поэтому данная среда выглядит одинаково на всех платформах, т.е. она не очень похожа на собственное приложение для каждой платформы. Некоторые усматривают в этом недостаток NetBeans и преимущество Eclipse.
- NetBeans имеет более скромный послужной список в качестве IDE для программирования на PHP, чем Eclipse, но быстро сравнилась с ней по своим функциональным возможностям — по крайней мере, в отношении самых простых задач разработки.
- На момент написания этой книги в NetBeans не поддерживались межплатформенные окончания строк. Так, если вы разрабатываете на платформе Windows и в ваших файлах требуются окончания строк для Linux, вам придется немного потрудиться, чтобы ввести окончания строк для Linux в файлы, созданные вами в Windows. В Eclipse (и многих других редакторах исходного текста программ) для этого достаточно настроить соответствующий параметр. Остается лишь надеяться, что такая возможность в скором времени появится и в NetBeans.

Если вы все еще не решились опробовать одну из этих сред, опробуйте обе, а затем выберите из них наиболее подходящую.

Коммерческие IDE

На рынке предлагается немало коммерчески доступных IDE для разработки программ на PHP. Ниже перечислены лишь некоторые из них.

- ZendStudio (`http://www.zend.com/en/products/studio`) — распространенная коммерческая IDE для разработки программ на PHP, основанная на Eclipse.
- Aptana Studio Pro (`http://www.aptana.com`) — другая коммерческая IDE для разработки программ на PHP, настоятельно рекомендуемая некоторыми разработчиками и также основанная на Eclipse.
- Komodo IDE (`http://www.activestate.com/komodo-ide`) — еще одна коммерческая IDE для разработки программ на PHP.
- PhpED от компании Nusphere (`http://www.nusphere.com/products/phped.htm`) — эта среда предоставляет встроенный отладчик и профилировщик, а также функции развертывания готового программного продукта.

Текстовые редакторы

Если материал предыдущего раздела не убедил вас остановить свой выбор на IDE или хотя бы опробовать ее, то в этом разделе рассматриваются более простые инструментальные средства для разработки программ на PHP.

Notepad++

Notepad++ представляет собой простой в использовании, но весьма эффективный открытый редактор исходного текста, предназначенный для Windows. В нем поддерживается выделение синтаксиса и отчасти завершение кода для целого ряда языков программирования, включая и PHP. Этот редактор исходного текста можно свободно загрузить по адресу `http://notepad-plus-plus.org`.

В качестве примера на рис. 2.26 приведен открытый в Notepad++ исходный файл программы на PHP, написанной для Joomla. Как следует из данного примера, в Notepad++ выделение ошибок не поддерживается.

Для пользователей Mac OS X имеется простой и удобный редактор исходного текста Smultron (`http://smultron.en.softonic.com/mac`), обеспечивающий выделение синтаксиса в коде PHP. А для пользователей Linux имеется стандартный системный редактор `gedit`, также обеспечивающий выделение синтаксиса в коде PHP.

Для написания программ на PHP можно также воспользоваться стандартным текстовым редактором вроде Notepad или vi, но рассчитывать на автоматическое выделение синтаксиса в таком редакторе не приходится. Любой редактор простого текста вполне подойдет для правки всех файлов Joomla. Очевидно, что чем дальше вы отступаете от функциональных возможностей IDE, тем труднее вам будет визуально проверять набранный исходный текст программы на наличие ошибок и опечаток.

Не менее важно понять, что для правки одного и того же набора файлов Joomla можно пользоваться самыми разными программными средствами. Даже если файл находится в

проекте, определенном в IDE, он по-прежнему доступен на уровне файловой системы для редактирования за пределами IDE. Таким образом, для решения различных задач разработки можно при желании пользоваться разными программными средствами.

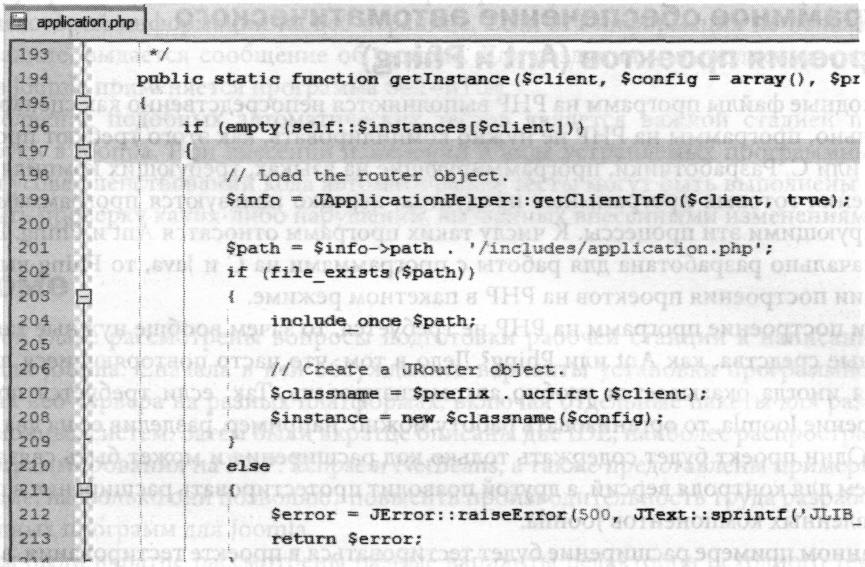


Рис. 2.26. Редактор исходного текста Notepad++

Другие инструментальные средства

Для разработки в Joomla могут пригодиться и другие программные средства. Ниже они рассматриваются вкратце.

Программное обеспечение контроля версий

Это программное обеспечение считается очень важным и просто необходимым для большинства программных проектов, включая и Joomla. Программное обеспечение контроля версий автоматически отслеживает каждое изменение в кодовой базе проекта, позволяя группе разработчиков вносить изменения в кодовую базу в организованном порядке. Если же изменение, внесенное в код, вызовет какие-нибудь затруднения, оно может быть отменено с целью устранить эти затруднения. Даже если вы разрабатываете проект в одиночку, программное обеспечение контроля версий все равно принесет вам огромную пользу. Обнаружив в проекте какую-нибудь ошибку, вы можете всегда вернуться назад, чтобы выяснить, где именно вкралась ошибка. Это нередко помогает сузить поиск ошибок и недостатков в проекте.

В проектах, разрабатываемых в системе управления контентом (CMS) и на платформе Joomla, для контроля версий применяется программа Git (`http://git-scm.com`). Подобные проекты размещаются на веб-сайте Github (`https://github.com/joomla`). К числу других распространенных программ контроля версий относятся Subversion (`http://subversion.apache.org`) и Mercurial (`http://mercurial.selenic.com`).

Если вы пользуетесь IDE, то средства контроля версий обычно включаются в саму IDE. Например, в Eclipse и NetBeans поддерживаются программы GIT, Subversion и Mercurial.

Программное обеспечение автоматического построения проектов (Ant и Phing)

Исходные файлы программ на PHP выполняются непосредственно как сценарии. Следовательно, программы на PHP не нужно компилировать, как этого требуют программы на Java или C. Разработчики, программирующие на языках, требующих компиляции или построения готовых программных продуктов, нередко пользуются программами, автоматизирующими эти процессы. К числу таких программ относятся Ant и Phing. Если Ant первоначально разработана для работы с программами на C и Java, то Phing выполняет операции построения проектов на PHP в пакетном режиме.

Если построение программ на PHP не требуется, то зачем вообще нужные такие программные средства, как Ant или Phing? Дело в том, что часто повторяющиеся пакетные задания иногда оказывается удобно автоматизировать. Так, если требуется построить расширение Joomla, то организовать работу можно, например, разделив ее на два проекта в IDE. Один проект будет содержать только код расширения и может быть связан с хранилищем для контроля версий, а другой позволит протестировать расширение в пределах установленных компонентов Joomla.

В данном примере расширение будет тестироваться в проекте тестирования, а изменения в коде — отслеживаться в проекте расширения. Возможно, было бы удобнее сначала изменить код в исходных файлах проекта тестирования, а после проверки на работоспособность скопировать файлы из проекта тестирования в проект расширения. С помощью программы Ant или Phing процесс копирования файлов из одного проекта в другой можно было бы автоматизировать. А кроме того, можно автоматизировать процесс создания архивов для упаковки расширений.

Учебный материал по применению Phing с подобной целью доступен на сайте документации на Joomla по адресу `http://docs.joomla.org/Setting_up_your_workstation_for_extension_development`.

Программы автоматизации тестирования

В разработке программного обеспечения наметилась устойчивая тенденция к автоматизации процесса тестирования программ. Автоматические тесты могут быть написаны один раз, а затем выполнены неоднократно для обнаружения ошибок, вкравшихся в кодовую базу при внесении в нее изменений.

В проекте Joomla в настоящее время применяются две разновидности автоматического тестирования: блочное и системное. Блочное тестирование служит для тестирования платформы Joomla, а для выполнения блочных тестов — программа на PHP под названием `PHPUnit`. Это дает возможность писать тесты, непосредственно снабжающие данными метод в коде PHP, и сравнивать полученные результаты с предполагаемыми. Если они отличаются, выдается сообщение об ошибке.

Системное тестирование служит для проверки работоспособности системы CMS с точки зрения пользователя. С этой целью система CMS выполняется в сеансе работы

с браузером под управлением тестовой программы. А в браузере выполняется последовательность команд ввода данных, включая операции с мышью и ввода с клавиатуры, имитирующие действия пользователя тестируемого программного обеспечения. Документ в окне браузера может быть автоматически проверен в системном тесте на предмет появления некоторой информации на веб-странице. Если эта информация не появляется на веб-странице, выдается сообщение об ошибке. Для создания и выполнения системных тестов в Joomla применяется программа `Selenium`.

Выполнение подобных автоматических тестов является важной стадией процесса разработки в Joomla. При внесении изменений в виде устраняемых программных ошибок или усовершенствований кода автоматические тесты могут быть выполнены с целью упростить проверку каких-либо нарушений, вызванных внесенными изменениями.

Резюме

В этой главе рассмотрены вопросы подготовки рабочей станции к написанию программ для Joomla. Сначала в ней обсуждались варианты установки программного обеспечения веб-сервера на разных платформах, включая отдельные пакеты для различных операционных систем. Затем были вкратце описаны две IDE, наиболее распространенные для программирования на PHP: Eclipse и NetBeans, а также представлены примеры, показывающие, насколько они позволяют повысить производительность труда разработчиков прикладных программ для Joomla.

Затем были вкратце рассмотрены разные варианты редакторов исходного текста для тех, кто не желает пользоваться IDE. И в конце главы было кратко описано программное обеспечение контроля версий (вроде Subversion и Mercurial), автоматического построения проектов (типа Phing и Ant), а также автоматического тестирования.

Итак, рабочую станцию можно считать подготовленной и настроенной, и теперь можно приступать непосредственно к написанию прикладных программ для Joomla.

3

Принцип действия Joomla!

Любите ли вы разбирать предметы на части, чтобы понять, как они устроены? Является ли веб-программирование новой для вас сферой деятельности, и хотите ли вы понять, чем оно отличается от других видов программирования? Желаете ли вы знать, как и почему Joomla! имеет именно такую организацию, а не иную? Если вы дадите утвердительные ответы на все эти вопросы, значит, эта глава написана именно для вас!

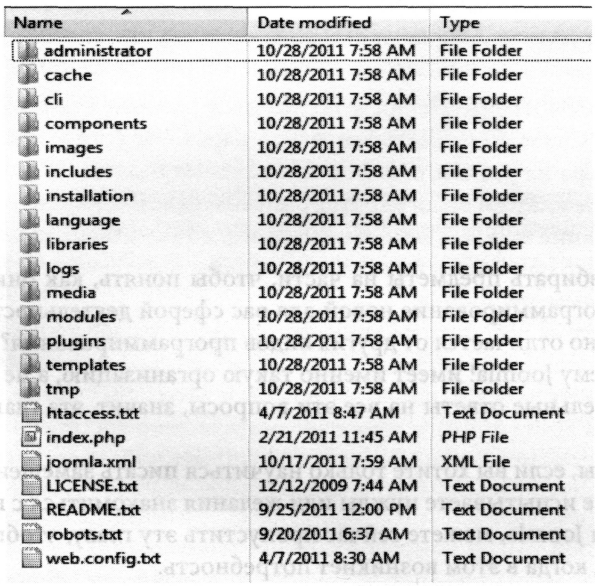
С другой стороны, если вы хотите только научиться писать замещения или расширения Joomla и не испытываете нужды или желания знакомиться с внутренним механизмом работы Joomla, можете сейчас пропустить эту главу, чтобы вернуться к ней впоследствии, когда в этом возникнет потребность.

В этой главе будет рассмотрен внутренний механизм работы Joomla и принцип его действия, чтобы выяснить, почему он работает именно так, а не иначе. По ходу изложения будут исследованы файлы и папки, образующие установленные компоненты Joomla, а также пояснено их назначение. После этого будет подробно описан сеанс работы Joomla, чтобы стало понятнее, что же происходит внутри механизма Joomla при заполнении форм, выборе кнопок или ссылок на страницах веб-сайта.

В этой главе будут также упомянуты места, где можно или даже нужно употребить специальные имена папок или файлов с написанными программами, чтобы они были найдены в Joomla и запущены на выполнение в нужный момент. Попутно будет показано, как получать доступ к часто требующимся объектам из программ, написанных для Joomla. И в конце этой главы будет дан краткий обзор таблиц базы данных Joomla.

Краткий обзор папок Joomla

При установке Joomla создаются папки и файлы, приведенные на рис. 3.1. После установки Joomla в корневой папке будет создан файл `configuration.php`. В рабочей версии папка `installation` обычно удаляется, поскольку ее наличие представляет уязвимость с точки зрения безопасности. Если вы загрузите Joomla из хранилища Github или SVN, то получите папку `tests`, содержащую файлы автоматических блочных и системных тестов, папку `docs` с инструкциями по составлению документации в формате PDF, а также папку `build` с программой построения архивных файлов Joomla. Эти папки удаляются, когда Joomla составляется в пакет для обычной установки.



Name	Date modified	Type
administrator	10/28/2011 7:58 AM	File Folder
cache	10/28/2011 7:58 AM	File Folder
cli	10/28/2011 7:58 AM	File Folder
components	10/28/2011 7:58 AM	File Folder
images	10/28/2011 7:58 AM	File Folder
includes	10/28/2011 7:58 AM	File Folder
installation	10/28/2011 7:58 AM	File Folder
language	10/28/2011 7:58 AM	File Folder
libraries	10/28/2011 7:58 AM	File Folder
logs	10/28/2011 7:58 AM	File Folder
media	10/28/2011 7:58 AM	File Folder
modules	10/28/2011 7:58 AM	File Folder
plugins	10/28/2011 7:58 AM	File Folder
templates	10/28/2011 7:58 AM	File Folder
tmp	10/28/2011 7:58 AM	File Folder
htaccess.txt	4/7/2011 8:47 AM	Text Document
index.php	2/21/2011 11:45 AM	PHP File
joomla.xml	10/17/2011 7:59 AM	XML File
LICENSE.txt	12/12/2009 7:44 AM	Text Document
README.txt	9/25/2011 12:00 PM	Text Document
robots.txt	9/20/2011 6:37 AM	Text Document
web.config.txt	4/7/2011 8:30 AM	Text Document

Рис. 3.1. Состав папок и файлов в версии Joomla 2.5

Пользовательская и административная части

Если вам приходилось раньше пользоваться Joomla, то понятия пользовательской (внешней или клиентской) и административной (внутренней или серверной) частей вебсайта должны быть вам уже знакомы. Их можно рассматривать как совершенно отдельные приложения, пользующиеся общими библиотеками.

Заглянув в подпапки, находящиеся в папке `administrator`, вы обнаружите немало тех же самых папок, что и в папке верхнего уровня. К их числу относятся папки `cache`, `components`, `includes`, `language`, `modules` и `templates`. В папке `administrator` находится также собственная версия файла `index.php`. Поэтому если ввести URL по вебадресу `http://mywebsite.org/administrator`, будет загружен файл `administrator/index.php`, запускающий администраторское приложение. А если ввести URL по вебадресу `http://mywebsite.org`, будет загружен файл `index.php` из папки верхнего уровня, запускающий пользовательское приложение.

Каждая из этих папок вкратце рассматривается далее. Но поскольку папка `administrator` имеет такую же структуру, как и папка верхнего уровня, она будет описана последней.

Папка `cache`

Назначение папки `cache` легко понять. Именно в ней сохраняются кешируемые файлы. Системе динамического веб-сайта приходится немало потрудиться, чтобы собрать всю необходимую информацию для отображения на странице. Так, если на странице отображается компоновка блога, система должна получить доступ к базе данных, чтобы извлечь из нее статьи для блога, разместить их на странице и загрузить модули, требующиеся для страницы.

На многих реальных веб-сайтах информация для страницы может оставаться неизменной целыми часами, а то и днями. Так, если 50 посетителей веб-сайта запрашивают одну и ту же страницу в течение часа, а информация на ней изменяется лишь один раз в день, то извлекать ее из базы данных пришлось бы 1200 раз (50×24), прежде чем она фактически изменилась бы.

Если же активизировать кеширование (т.е. буферизацию) данных, то можно попытаться сократить количество обращений к базе данных для получения одной и той же информации. Так, если кешировать страницу в рассматриваемом здесь примере, то ее копия будет сохранена в папке `cache` на заданный период времени, например 15 минут. Когда первый посетитель веб-сайта запросит эту страницу, она будет скомпонована из базы данных обычным образом, а ее копия — сохранена в папке `cache`.

А когда второй посетитель веб-сайта запросит ту же самую информацию, система сначала проверит, имеется ли кешированная копия страницы, срок действия которой еще не истек (в данном случае он должен быть меньше 15 минут). Если кешированная копия имеется, страница извлекается именно из этой копии, а следовательно, ее не нужно компоновать снова, обращаясь за информацией к базе данных. Благодаря этому страница загружается намного быстрее, а кроме того, сокращается количество операций, выполняемых сервером. На постоянно занятом веб-сайте, где частота просмотра выше частоты изменений в базе данных, кеширование позволяет значительно повысить общую производительность системы.

Файлы удаляются из папки `cache` по команде Site⇨Maintenance⇨Clear Cache (Сайт⇨Сопровождение⇨Очистить кеш) или Site⇨Maintenance⇨Purge Expired Cache (Сайт⇨Сопровождение⇨Очистить просроченный кеш). Если вам приходится эксплуатировать крупный рабочий сайт, то для удаления ненужных кешированных файлов из данной папки можете запустить на выполнение специальное задание на уровне системы.

Папка `cli`

Папка `cli` была введена в качестве стандартного места для хранения приложений с интерфейсом командной строки (CLI). Одной из самых замечательных особенностей платформы Joomla является возможность легко создавать приложения с интерфейсом командной строки, взаимодействующие с веб-сайтом Joomla. Такие приложения отлично подходят для выполнения по графику, например, с помощью имеющейся в Linux программы `cron`. Характерный тому пример будет продемонстрирован в главе 13.

Папка components

Компонент является главным элементом отдельной страницы и обычно соответствует пункту меню. При выборе пункта меню загружаемая страница содержит компонент, определяемый этим пунктом меню.

Если просмотреть подпапки с компонентами, то в них можно обнаружить каждый базовый компонент Joomla: com_banners, com_contact, com_content (для статей), com_finder (для интеллектуального поиска), com_mailto, com_media, com_newsfeeds, com_search, com_users, com_weblinks, а также com_wrappers. Нетрудно догадаться, что имена папок отдельных компонентов начинаются с префикса com_, после которого следует название конкретного компонента. Это один из примеров строгих условных обозначений, повсеместно принятых в Joomla. Подробнее о них речь пойдет далее в этой главе.

При создании нового пункта меню для выбора одного из основных компонентов имеется несколько вариантов. И все эти варианты приведены на рис. 3.2.

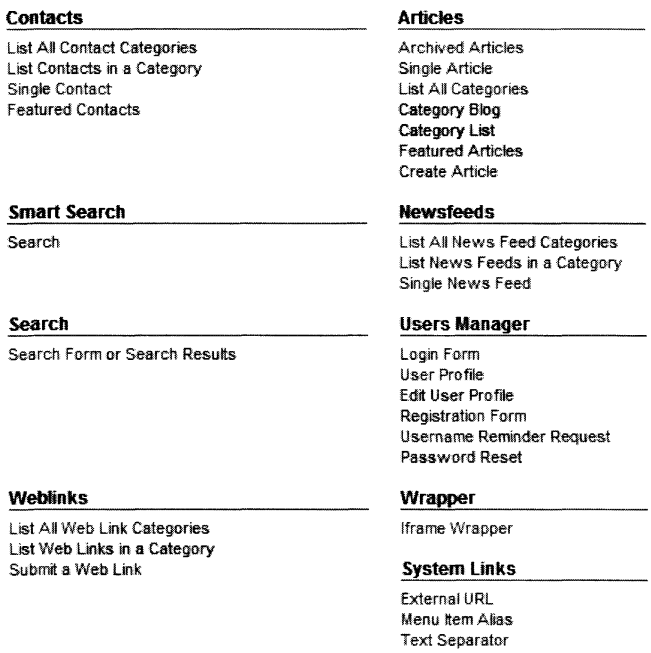


Рис. 3.2. Основные типы пунктов меню для пользовательской части

Большинство типов пунктов меню совпадают с названиями соответствующих папок. К их числу относятся контакты, статьи (из папки com_content), ленты новостей, поиск, пользователи, веб-ссылки и оболочка.

Компоненты com_banners, com_mailto и com_media не соответствуют типам пунктов меню. Крупные заголовки (так называемые баннеры) также являются компонентами, но они размещаются на странице внутри модулей. Если щелкнуть на пиктограмме E-mail, чтобы отправить кому-нибудь статью по электронной почте, будет вызван ком-

понент com_mailto. А компонент com_media вызывается при нажатии кнопки Image (Изображение) во время редактирования статьи в пользовательской части, чтобы загрузить компонент Media Manager (Диспетчер мультимедийного содержимого).

Эти компоненты не отображаются как основная часть страницы и поэтому не подпадают под простое определение компонента. И это приводит нас к следующему более подробному и формальному определению компонента: компонент представляет собой расширение, которое должно выполнять некоторое действие, помимо отображения чего-нибудь. Так, если посетитель веб-сайта щелкнет на баннере, последний должен зарегистрировать факт совершения щелчка в базе данных и, возможно, направить посетителя по другому URL. Аналогично, если посетитель веб-сайта щелкнет на пиктограмме E-mail, ему должна быть сначала предложена форма для заполнения, а затем отправлена электронная почта. Если же требуется вставить изображение в статью, для этой цели снова должны быть выполнены определенные действия, например, выгрузка изображения на веб-сервер. Таким образом, во всех подобных случаях компонент должен предпринимать определенные действия, хотя он может и не быть основным содержимым страницы.

Папка images

Папка images является одним из тех мест, где могут храниться файлы изображений, предназначенных для сайта. Например, в основном варианте распространения Joomla имеются подпапки banners и sampledata. Как и следовало ожидать, в папке banners хранятся изображения для образцов баннеров, включенных в основной вариант распространения Joomla. А в папке sampledata хранятся изображения, используемые в выборочных данных, которые можно дополнительно установить вместе с Joomla.

Важное примечание по поводу файлов index.html

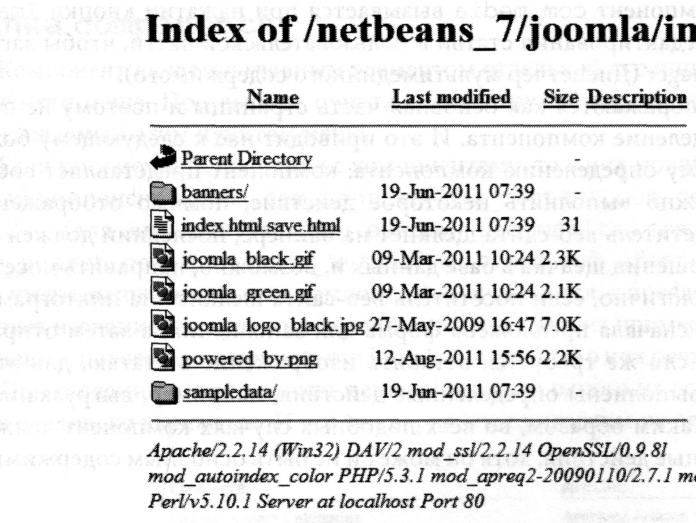
Нетрудно заметить, что каждая подпапка в основном варианте распространения Joomla (кроме папок administrator и installation, в которых находятся файлы index.php) содержит файл index.html. Если открыть этот файл в любом редакторе, то появится следующее его содержимое:

```
<!DOCTYPE html><title></title>
```

Этот HTML-файл содержит элемент разметки <html> и вложенный в него пустой элемент разметки <title>. В этом файле отсутствует какое-либо содержимое, и поэтому при его отображении в окне браузера появляется фактически пустая страница.

Зачем же подобные файлы присутствуют во всех подпапках? Это сделано из соображений безопасности. Так, если пользователь попытается просмотреть папку, веб-сервер осуществит поиск файла index.php или index.html. И если ему не удастся найти эти файлы, в окне браузера появится список файлов и подпапок, находящихся в данной папке.

Вы можете убедиться в этом сами, временно переименовав файл images/index.html на images/index1.html. Если затем вы попытаетесь просмотреть содержимое папки по URL <ваша_корневая_папка_Joomla!>/images, то увидите нечто, аналогичное приведенному на рис. 3.3. Если же вы переименуете файл обратно на index.html, то по тому же самому URL в окне браузера появится пустая страница.



Index of /netbeans_7/joomla/images

Name	Last modified	Size	Description
Parent Directory		-	
banners/	19-Jun-2011 07:39	-	
index.html.save.html	19-Jun-2011 07:39	31	
joomla_black.gif	09-Mar-2011 10:24	2.3K	
joomla_green.gif	09-Mar-2011 10:24	2.1K	
joomla_logo_black.jpg	27-May-2009 16:47	7.0K	
powered_by.png	12-Aug-2011 15:56	2.2K	
sampledata/	19-Jun-2011 07:39	-	

Apache/2.2.14 (Win32) DAV/2 mod_ssl/2.2.14 OpenSSL/0.9.8l mod_autoindex_color PHP/5.3.1 mod_apreq2-20090110/2.7.1 mod_perl/2.0.4 Perl/v5.10.1 Server at localhost Port 80

Рис. 3.3. Вид окна браузера с папкой `images` без файла `index.html`

Разрешение несанкционированному пользователю просматривать информацию о файлах и папках на веб-сервере считается рискованным с точки зрения безопасности. Всякий, пытающийся взломать веб-сайт, может воспользоваться этой информацией и затем прощупать веб-сайт на предмет уязвимости. Именно поэтому в каждой подпапке системы должен непременно присутствовать пустой файл `index.html`.

Папки `installation` и `administrator` являются единственными, где отсутствует файл `index.html`. Дело в том что в них находится файл `index.php`, который служит в качестве отправной точки для процесса установки и доступа к административной части веб-сайта. В данном случае файл `index.html` будет препятствовать загрузке файла `index.php`.

Папка `includes`

Как упоминалось ранее, из папки `administrator` осуществляется управление административной частью веб-сайта, которую можно считать приложением, совершенно отдельным от пользовательской части веб-сайта. Именно по этой причине, начиная сеанс работы в пользовательской или административной части, нужно загружать разные классы и константы в оперативную память.

Папка `includes` содержит программы, характерные для пользовательской или административной части как отдельного приложения. На верхнем уровне папки `includes` находятся файлы, которые требуется загрузить, чтобы начать сеанс работы в пользовательской части веб-сайта, тогда как в подпапке `administrator/includes` — файлы, которые требуется загрузить, чтобы начать сеанс работы в административной части. Файлы, содержащиеся в самой папке `includes`, приведены в табл. 3.1.

Таблица 3.1. Файлы, содержащиеся в корневой папке `includes`

Файл	Действие
`application.php`	Создает класс `JSite`
`defines.php`	Определяет константы для путей к файлам
`framework.php`	Загружает часто используемые части интегрированной среды Joomla, называемой также платформой
`menu.php`	Загружает все меню веб-сайта в оперативную память
`pathway.php`	Создает класс `JPathwaySite`, используемый в модуле `mod_breadcrumbs` для отображения пути к текущему пункту меню
`router.php`	Загружает класс `JRouterSite`. (Подробнее о маршрутизаторе речь пойдет далее в этой главе.)

Папка `installation`

В папке `installation` содержится совершенно отдельное мини-приложение, имеющее единственное назначение: первоначальную установку Joomla. Когда вы начинаете первый сеанс работы в пользовательской или административной части, система проверяет наличие файла `configuration.php` в корневой папке. Если она не обнаружит этот файл, то предположит, что вам нужно произвести установку, и поэтому загрузит установочное мини-приложение.

А поскольку это совершенно автономное мини-приложение, то у него имеется собственная папка `includes`, а также другие папки, требующиеся для выполнения процесса установки. К числу самых замечательных особенностей Joomla относится простота установки. Весь процесс установки совершается автоматически, включая и создание базы данных. И все это делает установочное мини-приложение.

Папка `language`

Локализация Joomla уже осуществлена на 68 языках мира, и их число будет только расти. Как пояснялось в главе 2, практически весь текст в Joomla переводится перед тем, как появиться в окне браузера. Это делается с помощью языковых пакетов, устанавливающих ряд файлов языковой поддержки. Все эти языковые пакеты устанавливаются в подпапке `languages`.

При стандартной установке на английском языке создаются подпапки `en-GB` и `overrides`. В первой из них хранятся все файлы языковой поддержки, требующиеся для перевода пользовательской части веб-сайта на выбранный язык мира (в данном случае британский английский, т.е. официальный вариант английского языка для Joomla). А вторая папка связана с новым средством, появившимся в версии 1.6: языковыми замещениями. Языковые замещения позволяют изменить содержимое любой текстовой строки, не меняя весь файл языковой поддержки. Характерный тому пример будет приведен в главе 4.

Следует также иметь в виду, что на каждое расширение зачастую приходится два файла языковой поддержки. Один из них имеет расширение `.sys.ini`, а другой — обычное расширение `.ini`. Файл с расширением `.sys.ini` загружается для получения наименования и описания расширения, например, для отображения в компоненте Module Manager (Диспетчер модулей). А файл с расширением `.ini` загружается во время работы с расширением для отображения всего текста, используемого в данном расширении.

Папка libraries

Библиотека представляет собой собрание программ, предназначенных для неоднократного применения в различных контекстах. Как правило, библиотека решает конкретную задачу или ряд связанных вместе задач. Библиотеки программ обычно хранятся в подпапках основной библиотечной папки.

В версии Joomla 1.5 использовалось 14 библиотек. Из них 13 библиотек были внешними, созданными сторонними разработчиками, а одна — внутренней библиотекой Joomla. А в версии Joomla 2.5 количество библиотек уменьшилось до четырех. Три из них являются внешними: библиотека `phpmailer` служит для отправки электронной почты, библиотека `phputf8` — для операций с текстом в кодировке UTF-8, а библиотека `simplepie` — для организации лент новостей типа RSS (или веб-каналов).

Наибольший интерес представляет четвертая библиотека, хранящаяся в папке `joomla`. В данной папке содержатся классы платформы Joomla, как поясняется более подробно далее в этой главе. В версии Joomla 2.5 появилась еще одна папка под названием `cms`. В ней находятся файлы библиотек, применяемые в системе управления информационным наполнением (CMS), но не на платформе Joomla.

Папка logs

Эта папка служит для хранения системных журналов, формируемых в Joomla. События регистрируются в них, исходя из определенных условий. Например, любое состояние ошибки будет зарегистрировано в файле `error.log`. Кроме того, некоторые события регистрируются в результате перехода в режим отладки.

Папка media

В папке `media` хранятся вложенные таблицы стилей CSS, сценарии на языке Javascript, а также файлы изображений для некоторых компонентов, модулей и редакторов. Среди прочего, в подпапке `system` находится интегрированная среда JavaScript под названием MooTool, применяемая в Joomla для обеспечения некоторых функциональных возможностей JavaScript. Подробнее о JavaScript вообще и MooTools в частности речь пойдет в главе 12.

Папка modules

В папке `modules` находятся подпапки для каждого имеющегося модуля пользовательской части веб-сайта. На рис. 3.4, *слева*, приведены подпапки отдельных модулей, а на рис. 3.4, *справа*, — список типов модулей, появляющийся при создании нового модуля в диспетчере модулей. Для каждого типа модуля выделена отдельная подпапка. В табл. 3.2 перечислены наименования модулей и соответствующие им наименования подпапок.

При установке расширений модулей они добавляются в виде подпапок в папку `modules` и становятся полностью интегрированными в приложение Joomla. Более подробно модули рассматриваются в главе 6.



Рис. 3.4. Подпапки и типы модулей

Таблица 3.2. Типы модулей и соответствующие подпапки

Наименование подпапки	Наименование типа модуля	Перевод	Примечание
`mod_articles_archive`	Archived Articles	Архивные статьи	Называлась `mod_archive` в версии 1.5
`mod_articles_categories`	Articles Categories	Категории статей	Добавлена в версии 1.6
`mod_articles_category`	Articles Category	Категория статьи	Добавлена в версии 1.6 на замену папке `mod_sections`
`mod_articles_latest`	Latest News	Последние новости	Называлась `mod_latestnews` в версии 1.5
`mod_articles_news`	Articles – Newsflash	Статьи — Короткое экстренное сообщение	Называлась `mod_newsflash` в версии 1.5
`mod_articles_popular`	Most Read Content	Последнее прочитанное содержимое	Называлась `mod_mostread` в версии 1.5
`mod_banners`	Banners	Баннеры	

Окончание табл. 3.2

Наименование подпапки	Наименование типа модуля	Перевод	Примечание
mod_breadcrumbs	Breadcrumbs	Навигационные цепочки	
mod_custom	Custom HTML	Специальная HTML-разметка	
mod_feed	Feed Display	Отображение лентой	
mod_finder	Smart Search	Интеллектуальный поиск	Добавлена в версии 2.5
mod_footer	Footer	Нижний колонтитул	
mod_languages	Language Switcher	Переключатель языков	Добавлена в версии 1.6
mod_login	Login	Регистрация	
mod_menu	Menu	Меню	Называлась mod_mainmenu в версии 1.5
mod_random_image	Random Image	Произвольное изображение	
mod_related_items	Articles-Related Articles	Статьи — Связанные статьи	
mod_search	Search	Поиск	
mod_stats	Statistics	Статистика	
mod_syndicate	Syndication Feeds	Объединенные ленты подачи информации	
mod_users_latest	Latest Users	Последние пользователи	Добавлена в версии 1.6
mod_weblinks	Weblinks	Веб-ссылки	Добавлена в версии 1.6
mod_whosonline	Who's Online	Кто в Сети	
mod_wrapper	Wrapper	Оболочка	

Папка plugins

В папке plugins содержатся расширения подключаемых модулей. В версии 2.5 имеются десять типов подключаемых модулей, перечисленных в табл. 3.3 по отдельным подпапкам, где они находятся. Каждый тип подключаемого модуля отвечает за определенный вид действий на веб-сайте и реагирует на события, наступающие во время этих действий. Подробнее о подключаемых модулях речь пойдет в главе 5.

Таблица 3.3. Типы подключаемых модулей

Папка	Применение
authentication	При регистрации для проверки имени пользователя и пароля
captcha	При обработке экрана контрольного изображения
content	При создании и редактировании содержимого
editors	При загрузке стандартных редакторов (например, TinyMCE)

Окончание табл. 3.3

editors-xtd	При загрузке редакторов для создания кнопок типа Article (Статья), Image (Изображение), Page Break (Перейти на новую страницу), Read More (Подробнее)
extension	При установке, обновлении или удалении расширений
finder	При выполнении интеллектуального поиска
search	При выполнении обычного поиска
system	В различные моменты в течение цикла выполнения команд, включая начало сеанса работы, направление запроса, диспетчеризацию и воспроизведение. Более подробно об этом речь пойдет далее, в разделе "Внутреннее строение цикла выполнения команд в Joomla"
user	При создании, редактировании или удалении пользовательских данных и входе или выходе пользователей из системы

Папка templates

В Joomla содержимое каждой веб-страницы отделяется в как можно большей степени от ее представления на экране. А представление страницы находится под управлением расширений, называемых шаблонами.

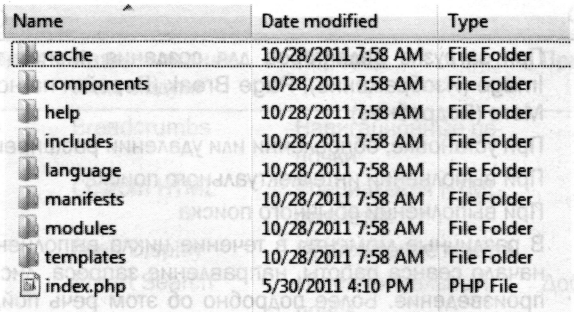
В папке templates каждый шаблон пользовательской части располагается в отдельной подпапке после установки в системе. При стандартной установке создаются подпапки atomic, beez_20 и beez5. Кроме того, в данную папку входит подпапка system. В ней находится системный шаблон с некоторыми данными компоновки, совместно используемыми всеми остальными шаблонами и самой системой в качестве резерва на тот случай, если класс или файл CSS не удастся найти в именованном шаблоне.

Папка tmp

Нетрудно догадаться, что папка tmp служит для временного хранения файлов в системе. Чаще всего эта папка используется при установке расширений. Так, если устанавливается новое расширение, копия его архивного файла сначала переносится в папку tmp, а затем распаковывается в нужные папки Joomla.

Папка administrator

Как упоминалось ранее, Joomla можно условно разделить на два отдельных приложения, совместно использующих некоторый общий код. Этими приложениями являются пользовательская и административная части веб-сайта. И это становится совершенно очевидно при просмотре подпапок папки administrator. В ней находится немало тех же самых подпапок, что и в папке верхнего уровня, в том числе подпапки cache, components, help, includes, language, modules, templates, а также файл index.php, как показано на рис. 3.5. Ниже приводится краткое описание всех этих подпапок.



Name	Date modified	Type
cache	10/28/2011 7:58 AM	File Folder
components	10/28/2011 7:58 AM	File Folder
help	10/28/2011 7:58 AM	File Folder
includes	10/28/2011 7:58 AM	File Folder
language	10/28/2011 7:58 AM	File Folder
manifests	10/28/2011 7:58 AM	File Folder
modules	10/28/2011 7:58 AM	File Folder
templates	10/28/2011 7:58 AM	File Folder
index.php	5/30/2011 4:10 PM	PHP File

Рис. 3.5. Содержимое папки `administrator`

Подпапка `administrator/cache`

Эта подпапка служит для хранения файлов, кешируемых в административной части веб-сайта. При кешировании страниц из административной части временно кешируемые файлы сохраняются именно в этой подпапке.

Подпапка `administrator/components`

В этой подпапке содержатся все компоненты, имеющиеся в административной части веб-сайта. Они обеспечивают по отдельности выбор каждого пункта меню в административной части. Все компоненты и соответствующие пункты меню из административной части перечислены в табл. 3.4.

Таблица 3.4. Компоненты административной части веб-сайта

Наименование подпапки	Пункты, выбираемые из меню в административной части	Перевод
com_admin	Help⇨Joomla! Help, Site⇨My Profile, а также Site⇨System Information	Справка⇨Справка по Joomla!, Сайт⇨Мой профиль, Сайт⇨Сведения о системе
com_banners	Components⇨Banners (варианты выбора)	Компоненты⇨Баннеры
com_cache	Site⇨Maintenance⇨Clear Cache, Site⇨Maintenance⇨Purge Expired Cache	Сайт⇨Очистить кеш, Сайт⇨Сопровождение⇨Очистить просроченный кеш
com_categories	Content⇨Category Manager, Components⇨Banners⇨Categories, Components⇨Contacts⇨Categories, Components⇨Newsfeeds Categories, Components⇨Weblinks⇨Categories	Содержимое⇨Диспетчер категорий, Компоненты⇨Баннеры⇨Категории, Компоненты⇨Контакты⇨Категории, Компоненты⇨Ленты новостей⇨Категории, Компоненты⇨Веб-ссылки⇨Категории
com_checkin	Site⇨Maintenance⇨Global Check-in	Сайт⇨Сопровождение⇨Глобальная регистрация
com_config	Site⇨Global Configuration	Сайт⇨Глобальная настройка
com_contact	Components⇨Contacts (варианты выбора)	Компоненты⇨Контакты

Окончание табл. 3.4

Наименование подпапки	Пункты, выбираемые из меню в административной части	Перевод
com_content	Content⇨Article Manager, Content⇨Featured Articles	Содержимое⇨Диспетчер статей⇨Избранные статьи
com_cpanel	Site⇨Control Panel	Сайт⇨Панель управления
com_finder	Components⇨Smart Search*	Компоненты⇨Интеллектуальный поиск
com_installer	Extension⇨Extension Manager	Расширение⇨Диспетчер расширений
com_languages	Extensions⇨Language Manager	Расширение⇨Диспетчер языков
com_login	Administrative Login (экран), Site⇨Logout	Регистрация администратора, Сайт⇨Выход из системы
com_media	Content⇨Media Manager	Содержимое⇨Диспетчер мультимедийного содержимого
com_menus	Menus⇨Menu Manager	Меню⇨Диспетчер меню (в меню Menus перечисляются варианты выбора из каждого меню, определенного для пользовательской части веб-сайта)
com_messages	Components⇨Messaging	Компоненты⇨Обмен сообщениями
com_modules	Extensions⇨Module Manager	Расширения⇨Диспетчер модулей
com_newsfeeds	Components⇨Newsfeeds	Компоненты⇨Ленты новостей
com_plugins	Extensions⇨Plugin Manager	Расширения⇨Диспетчер подключаемых модулей
com_redirect	Components⇨Redirect**	Компоненты⇨Переадресовать
com_search	Components⇨Search	Компоненты⇨Поиск
com_templates	Extensions⇨Template Manager	Расширения⇨Диспетчер шаблонов
com_users	Users (меню с вариантами выбора: User Manager, Groups, Access Levels, Mass Mail Users)	Пользователи (Диспетчер пользователей, Группы, Уровни доступа, Пользователи массовой отправки почты)
com_weblinks	Components⇨Weblinks	Компоненты⇨Вебссылки

** Добавлен в версии 2.5*
*** Добавлен в версии 1.6*

Подпапка administrator/help

В этой подпапке находится файл helpsites.xml, содержащий список доступных через Интернет справочных веб-сайтов. Как правило, при выборе кнопки Help (Справка) устанавливается связь с оперативной справкой по адресу docs.joomla.org.

Здесь же находится папка en-GB с HTML-файлом на британском английском языке по каждой теме справки. Благодаря этому появляется возможность разместить справочный файл на локальном сервере. При загрузке файлов из этой папки пользователь просто направляется к оперативно доступной справке. Но при желании эти файлы могут быть заменены справочными файлами, предоставляемыми локально.

Подпапка administrator/includes

Эта подпапка служит той же цели, что и рассмотренная выше папка includes верхнего уровня. В ней находятся файлы, требующиеся для того, чтобы начать сеанс работы в административной части веб-сайта. Все эти файлы перечислены в табл. 3.5.

Таблица 3.5. Файлы, находящиеся в подпапке administrator/includes

Файл	Назначение
application.php	Создает класс JAdministrator
defines.php	Определяет константы для путей к файлам
framework.php	Загружает наиболее употребительные части интегрированной среды Joomla, называемой также платформой
helper.php	Проверяет факт регистрации пользователя и передает ему требуемый вариант выбора компонента
menu.php	Загружает все меню администратора в оперативную память
router.php	Загружает класс JRouterSite. Более подробно маршрутизатор рассматривается далее в этой главе
toolbar.php	Предоставляет методы для загрузки панелей инструментов из административной части веб-сайта

Подпапка administrator/language

Эта подпапка служит той же цели, что и рассмотренная выше папка language верхнего уровня, за исключением того, что в ней предоставляются необходимые средства для перевода административной части веб-сайта на выбранный язык мира. Для каждого компонента, модуля, шаблона и подключаемого модуля в этой подпапке имеются свои файлы языковой поддержки.

Следует также иметь в виду, что, как и в пользовательской части веб-сайта, для каждого расширения в административной части также имеется один файл с расширением .sys.ini, а другой — с расширением .ini. Как пояснялось ранее, файл с расширением .sys.ini, как правило, содержит лишь наименование и описание соответствующего расширения для отображения в списке имеющихся расширений. А файл с расширением .ini служит для перевода всех текстовых строк, используемых в расширении.

Подпапка administrator/manifests

Эта подпапка появилась в Joomla в версии 1.6 для поддержки нового средства обновления одним щелчком кнопкой мыши. Она содержит следующие три подпапки: files, libraries и packages.

- В папке files находится файл joomla.xml, в котором перечислены все папки верхнего уровня и файлы из стандартной установки Joomla. В ней также содержится список серверов обновления (в виде URL), где можно найти обновленные версии этих файлов. Этот XML-файл сообщает средству обновления, какие именно файлы приложения следует обновить.
- В папке libraries находятся XML-файлы для каждой библиотеки, используемой в Joomla. В этих XML-файлах перечислены все папки и подпапки из библиотеки, а также URL того сервера, где можно проверить наличие обновлений для библиотеки.

- В папке `packages` также находится файл `joomla.xml`. Он служит для поддержки метода, выполняющего обновление установки Joomla в виде пакета, а не набора файлов. Такая возможность пока еще не реализована в версии 2.5, но может появиться в последующей версии Joomla. Она предполагает поддержку отношений зависимостей. Так, если запросить обновление или установку расширения A, для нормальной работы которого требуются расширения B и C, последние будут автоматически включены в процесс обновления или установки первого.

Подробнее об установке обновлений, выполняемых одним щелчком кнопкой мыши, можно узнать по адресу `http://joomlaprogrammingbook.com`.

Подпапка `administrator/modules`

Эта подпапка служит в административной части веб-сайта той же цели, что и папка `modules` верхнего уровня в пользовательской части. В ней находятся подпапки для каждого модуля из административной части. Все эти подпапки и соответствующие им модули перечислены в табл. 3.6. А на рис. 3.6 приведены сами модули административной части веб-сайта.

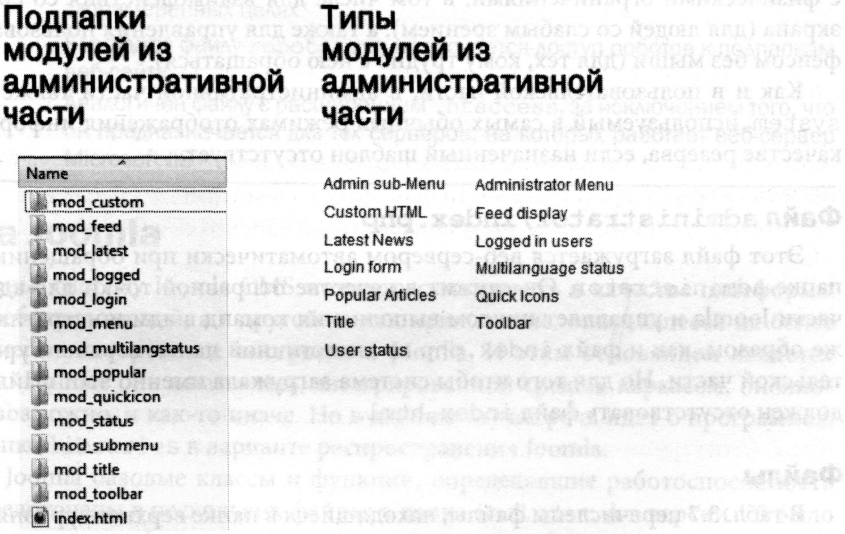


Рис. 3.6. Модули административной части веб-сайта

Таблица 3.6. Модули административной части веб-сайта и соответствующие им подпапки

Подпапка	Тип модуля	Перевод
`mod_custom`	Custom HTML	Специальная HTML-разметка
`mod_feed`	Feed Display	Отображение лентой
`mod_latest`	Latest News	Последние новости
`mod_logged`	Logged-In Users	Зарегистрировавшиеся пользователи
`mod_login`	Login	Регистрация

Окончание табл. 3.6

Подпапка	Тип модуля	Перевод
mod_menu	Menu	Меню
mod_multilangstatus	Multilanguage status	Состояние многоязычности
mod_popular	Popular Articles	Популярные статьи
mod_quickicon	Quick Icons	Пиктограммы оперативного доступа
mod_status	User Status	Состояние пользователя
mod_submenu	Admin Submenu	Подменю администрирования
mod_title	Title	Заглавие
mod_toolbar	Toolbar	Панель инструментов

Подпапка administrator/templates

В этой папке хранятся шаблоны, используемые для отображения страниц в административной части веб-сайта. В стандартную реализацию Joomla входят следующие два шаблона: Bluestork и Hathor. Шаблон Bluestork выбирается по умолчанию, тогда как шаблон Hathor служит для предоставления специальных возможностей пользователям с физическими ограничениями, в том числе для взаимодействия со средствами чтения экрана (для людей со слабым зрением), а также для управления пользовательским интерфейсом без мыши (для тех, кому трудно с нею обращаться).

Как и в пользовательской части, в административной части также имеется шаблон system, используемый в самых обычных режимах отображения информации, а также в качестве резерва, если назначенный шаблон отсутствует.

Файл administrator/index.php

Этот файл загружается веб-сервером автоматически при обращении браузера к подпапке administrator. Он служит в качестве отправной точки для административной части Joomla и управляет циклом выполнения команд в административной части таким же образом, как и файл index.php из аналогичной папки верхнего уровня в пользовательской части. Но для того чтобы система загружала именно этот файл, в данном месте должен отсутствовать файл index.html.

Файлы

В табл. 3.7 перечислены файлы, находящиеся в папке верхнего уровня Joomla. В таблице поясняется также их назначение.

Таблица 3.7. Файлы из папки верхнего уровня и их назначение

Файл	Назначение
configuration.php	Создается после успешной установки и содержит характерные для веб-сайта настройки, в том числе сведения о подключении к базе данных. Большинство, но не все эти настройки подлежат правке на экране глобальной настройки в административной части
htaccess.txt	Может быть переименован в файл с расширением .htaccess и используется вместе со средством mod_rewrite веб-сервера Apache. Если используется, то удаляет файл index.php из всех URL в Joomla, обеспечивая дополнительные меры безопасности

Окончание табл. 3.7

Файл	Назначение
index.php	Запускает процесс загрузки всех страниц пользовательской части веб-сайта Joomla. Загружается веб-сервером автоматически при загрузке по любому URL, указывающему на пользовательскую часть веб-сайта, т.е. не на папку administrator
joomla.xml	Является копией файла administrator/manifests/files/joomla.xml и используется во время установки, после чего удаляется из корневой папки
LICENSE.txt	Содержит текст лицензионного соглашения на Joomla, составленного в соответствии с Открытым лицензионным соглашением GNU (GNU GENERAL PUBLIC LICENSE) версии 2 от июля 1991 года
README.txt	Содержит самые общие сведения о Joomla
robots.txt	Веб-роботами называются программы, автоматически отслеживающие и собирающие сведения о веб-сайтах. В таких поисковых механизмах, как Google, роботы используются для обновления их индексов. Рассылающие нежелательную почту и прочие недобросовестные личности пользуются роботами для сбора адресов электронной почты и в других злонамеренных целях Благодаря файлу robots.txt исключается доступ роботов к подпапкам веб-сайта
web.config.txt	Аналогичен файлу с расширением .htaccess, за исключением того, что он предназначается для тех серверов, на которых работает веб-сервер Microsoft IIS

Платформа Joomla

С одной стороны, систему Joomla CMS можно рассматривать в качестве платформы для построения веб-приложений, а с другой — как приложение, опирающееся на более прочное основание, на котором все построено в Joomla. И этим основанием является платформа Joomla. Раньше она называлась интегрированной средой, каркасом, библиотеками Joomla, а возможно, и как-то иначе. Но в данном случае речь идет о программах, находящихся в папке libraries в варианте распространения Joomla.

До версии 1.5 Joomla базовые классы и функции, определявшие работоспособность Joomla, были сосредоточены в нескольких файлах в папке includes. А в версии 1.5 было решено сделать базовый код более четко организованным, чтобы упростить его сопровождение. При этом преследовалась цель сделать проект Joomla в большей степени похожим на профессиональную разработку программного обеспечения, опирающуюся на структуры, обеспечивающие базовую логику. В частности, создателям Joomla хотелось сделать разработку в этой среде более удобной для тех, у кого имеется предыдущий опыт программирования на Java или на платформе .NET. В силу этих причин и было решено построить то, что теперь называется платформой Joomla.

Если заглянуть в папку libraries, то в ней можно обнаружить пять папок и четыре исходных файла PHP (наряду с обязательным HTML-файлом). В файле loader.php содержится очень важный класс JLoader, автоматически включающий нужные программы из платформы, чтобы ими можно было воспользоваться в разрабатываемых проектах, а также две не менее важные функции jexit() и jimport().

Функция `jexit()` служит оболочкой для функции `exit`, фактически являющейся языковой конструкцией в PHP. Благодаря ей упрощается блочное тестирование, и поэтому ее следует непременно использовать в сценариях вместо функции `exit()`. А функция `jimport()` обеспечивает более короткий способ вызова метода `JLoader::import()` с синтаксисом, аналогичным оператору `import` в Java или оператору `using` в C#. Общий замысел состоит в том, чтобы предоставить функции `jimport()` в качестве параметра разделенный точками путь к библиотеке, чтобы воспользоваться ею, а платформа сама определит, насколько эта библиотека доступна для использования.

В папке `libraries` содержатся папки `joomla`, `cms`, `phpmailer`, `phputf8` и `simplepie`. Первые три из них не относятся непосредственно к Joomla и содержат сторонние библиотечные пакеты, используемые в системе Joomla CMS для обеспечения различных функциональных возможностей.

- Папка `phpmailer` содержит пакет, написанный на PHP и предназначенный для обработки сложных отправлений электронной почты.
- Папка `phputf8` содержит пакет, упрощающий взаимодействие PHP с кодировкой UTF-8, расширяя собственные функциональные возможности PHP для обработки символьных строк в коде UTF-8.
- Папка `simplepie` содержит пакет, предназначенный для синтаксического анализа информации, подаваемой лентой в формате XML, например в веб-каналах ATOM и RSS.

Папка `cms` была добавлена в версии 2.5. В ней содержатся классы из библиотеки Joomla, требующиеся для нормальной работы системы Joomla CMS, но не применяемые на платформе Joomla.

В папке `libraries/joomla` находятся следующие два файла.

- Файл `factory.php` содержит статический класс `JFactory` и предоставляет быстрый способ получения различных системных объектов из общего интерфейса.
- Файл `methods.php` содержит ряд статических вспомогательных классов для взаимодействия с URL и перевода текста.

Папки, находящиеся в папке `libraries/joomla`, называются библиотечными пакетами. Они сгруппированы в логическом порядке по видам функций, которые обеспечивают эти пакеты, как показано в табл. 3.8.

Таблица 3.8. Библиотечные пакеты платформы Joomla

Библиотечный пакет	Описание
`access`	Функции для системы списков контроля доступа (ACL), что позволяет контролировать различные действия, которые пользователи могут выполнять над объектами на веб-сайте
`application`	Функции для расширений и класса основного приложения
`base`	Классы самого низкого уровня, обычно являющиеся абстрактными и реализующие определенного рода шаблон проектирования или структуру данных
`cache`	Классы, реализующие различные виды кеширования и механизмы сохранения данных

Окончание табл. 3.8

Библиотечный пакет	Описание
client	Клиентские классы для связывания с различными серверами, например FTP или LDAP
database	Классы для связывания с базой данных и взаимодействия с ней
document	Классы для составления документа, отправляемого в конечном итоге браузеру, и манипулирования этим документом
environment	Классы для обработки запросов типа GET, POST, COOKIE и заголовков ответов
error	Классы, помогающие обрабатывать ошибки
event	Классы, составляющие систему событий для их инициирования, реагирования и обработки
filesystem	Классы для взаимодействия с файловой системой
filter	Классы, дополнительно обрабатывающие и отфильтровывающие вводимые и выводимые значения
form	Классы, помогающие составлять, обрабатывать веб-формы и манипулировать ними
github	Классы для взаимодействия с хранилищем в системе контроля версий платформы по адресу http://github.com
html	Вспомогательные методы для формирования разметки HTML и упрощения загрузки встроенных таблиц стилей CSS и сценариев Javascript
http	Классы для обработки различных видов запросов по протоколу HTTP
image	Классы для обработки файлов изображений
installer	Классы для установки и обновления расширений приложения
language	Классы для перевода символьных строк на разные языки мира
log	Классы для реализации функций регистрации на платформе
mail	Вспомогательные методы для отправки электронной почты
plugin	Классы, связанные с системой подключаемых модулей
registry	Классы для работы со сложными объектами данных, например, для импорта и экспорта данных в форматах INI, JSON и XML
session	Классы для создания, сохранения сеанса работы и манипулирования им
string	Классы для обработки символьных строк
updater	Классы для поддержки автоматически обновляющихся расширений
user	Классы для взаимодействия с пользовательскими объектами или манипулирования ими
utilities	Вспомогательные классы, не вошедшие в отдельный пакет

Веб-программирование в сравнении с обычным программированием

Динамическое веб-программирование, аналогичное для Joomla, отличается от обычного программирования, например, настольных приложений, двумя важными аспектами. Оба эти аспекта связаны с тем, как сохраняется состояние программы и какой вид проверки достоверности команд требуется.

Сохранение состояния программы

Первое отличие веб-программирования состоит в том, каким образом состояние программы сохраняется во время ее выполнения. Под состоянием в данном случае подразумевается, что именно ей известно о себе самой и ее окружении из того, что сохраняется в оперативной памяти компьютера. Состояние программы можно рассматривать как своего рода "самосознание" программы, т.е. что она собой представляет и что с ней происходит. Сравним для примера работу настольной программы электронных таблиц с Joomla в отношении того, как они сохраняют свое состояние.

В качестве аналогии представим обе рассматриваемые здесь программы в виде Аладдина и его волшебной лампы. Допустим, что когда мы выдаем программе команду, на компьютере Аладдина ее выполняет некий программный джин. Если джин электронных таблиц рад каждый раз выполнить сколько угодно наших желаний, то джин Joomla – только одно.

Для загрузки программы электронных таблиц в оперативную память нам нужно сначала щелкнуть на ее пиктограмме, расположенной на рабочем столе операционной системы. Аладдин отреагирует на наши действия и потрет свою волшебную лампу, чтобы появился джин. Как только он появится из лампы, Аладдин прикажет ему: "Отобрази программу электронных таблиц на экране!" Джин выполнит это приказание и станет ждать следующего приказания Аладдина.

Далее нам потребуется открыть файл составления бюджета. Наше желание Аладдин передаст джину, и файл откроется. Этот процесс может продолжаться до тех пор, пока мы не закроем программу. И тогда Аладдин прикажет джину: "Ты больше не нужен. Возвращайся обратно в лампу!", после чего джин электронных таблиц исчезнет внутри волшебной лампы.

А теперь посмотрим, как Аладдин будет обращаться с джином Joomla. Начнем с загрузки начальной страницы в окне браузера по заданному URL. Аладдин отреагирует на наши действия и потрет свою волшебную лампу. Как только появится джин Joomla, Аладдин прикажет ему: "Отобрази страницу по ее URL!" И джин, как по волшебству, выведет нужную нам начальную страницу в окне браузера. Но поскольку джин Joomla выполняет каждый раз только одно приказание, то он сразу же исчезнет внутри волшебной лампы!

Далее щелкнем на пункте меню в верхней части начальной страницы, чтобы отобразить статью о парках. Аладдину снова придется потереть свою волшебную лампу, чтобы появился джин, а затем приказать ему: "Загрузи статью о парках!" Джин послушно загрузит новую страницу и сразу же исчезнет внутри волшебной лампы.

Этот процесс будет продолжаться до тех пор, пока мы не закроем окно браузера или не покинем сайт Joomla. В этот момент Аладдину *вообще не нужно ничего делать, поскольку джин уже вернулся в лампу.* И это очень важное обстоятельство: в веб-программировании нельзя полагаться на то, что пользователь сам закроет программу. Правда, этого и не требуется. Рассмотренные выше действия в обеих программах из данного примера поэтапно представлены в табл. 3.9 и 3.10. Всякий раз, когда в такой веб-программе, как Joomla, пользователь выбирает ссылку или кнопку предъявления заполненной формы щелчком кнопкой мыши, начинается так называемый цикл выполнения запросов или команд. Любые данные формы, URL и все, что связано с запросом, оформляется браузером в пакет и отправляется веб-серверу.

Таблица 3.9. Последовательность приказаний, выполняемых джином электронных таблиц

Пользователь	Аладдин	Джин электронных таблиц
Щелкает на пиктограмме программы электронных таблиц	Потирает волшебную лампу и приказывает джину: "Открой программу электронных таблиц!"	Появляется из лампы, открывает программу электронных таблиц и ждет следующего приказания
Выбирает файл составления бюджета, чтобы открыть его	Приказывает джину: "Открой файл составления бюджета!"	Открывает файл и ждет следующего приказания
Отдает другие приказания	Передает приказания джину	Выполняет каждое приказание и ждет следующего
Выбирает команду выхода из программы	Приказывает джину: "Закрой программу и вернись обратно в лампу!"	Закрывает программу и возвращается в лампу

Таблица 3.10. Последовательность приказаний, выполняемых джином Joomla

Пользователь	Аладдин	Джин Joomla
Вводит URL начальной страницы в окне браузера	Потирает волшебную лампу и приказывает джину: "Загрузи страницу по ее URL!"	Появляется из лампы, отображает страницу по ее URL в окне браузера и возвращается в лампу
Щелкает на ссылке на статью о парках	Потирает волшебную лампу и приказывает джину: "Открой статью о парках!"	Появляется из лампы, открывает файл статьи и возвращается в лампу
Отдает другие приказания	Потирает волшебную лампу и передает каждое приказание джину по отдельности	Появляется из лампы, выполняет каждое приказание по отдельности, возвращаясь всякий раз в лампу
Закрывает браузер	Никаких действий не требуется	Никаких действий не требуется

В промежутках между циклами выполнения запросов, формируемых в Joomla (и любой другой веб-программе), в оперативной памяти компьютера ничего не запоминается. Каждый цикл должен начинаться с самого начала для создания всех объектов в программе. Всякий раз джин Joomla начинает свое служение сызнова.

В таком случае возникает вопрос: каким образом джин Joomla запоминает важные сведения в промежутках между последовательными циклами выполнения запросов? В частности, ему нужно знать, какой пользователь к нему обращается и какие действия этому пользователю разрешено выполнять. Но если память джина совершенно пуста в начале каждого цикла выполнения запросов, то как он справляется со своей задачей?

Данные в промежутках между циклами выполнения запросов можно сохранять разными способами. Чаще всего для этой цели используется переменная сеанса, которая поддерживается на сервере для каждого пользователя в данном сеансе работы. Она сохраняется на диске и доступна для Joomla. Как правило, файл сеанса автоматически удаляется или становится недоступным после определенного периода бездействия (например, в течение 15 минут). Так, джин Joomla может выяснить из файла сеанса, какой именно пользователь

к нему обращается, не требуя от него регистрации всякий раз. Он может также "запомнить", где находился пользователь в последнем цикле выполнения команд и какие параметры он настроил (например, порядок сортировки столбцов на экране).

Еще один способ сохранения данных в промежутках между циклами выполнения запросов состоит в обращении к базе данных. Она обновляется по мере внесения изменений на веб-сайте, например, при добавлении статей или других составляющих компонентов или же при изменении профиля пользователя. В результате обращения к базе данных в последующих циклах появляется обновленная информация.

Сеанс работы или база данных позволяет Joomla найти информацию из предыдущих циклов выполнения запросов или команд, а пользователь воспринимает различные циклы выполнения команд как непрерывный ход выполнения программы. Следует, однако, иметь в виду, что каждый цикл выполнения запросов должен быть автономным. Это имеет очень важные последствия для написания кода.

Контроль и проверка команд

Отличия между двумя рассматриваемыми здесь видами программирования имеют большое значение и с точки зрения безопасности. В автономной настольной прикладной программе все возможные команды, как правило, определены заранее. Они чаще всего выбираются щелчком кнопкой мыши из списка или меню. Даже если команды набираются вручную, они обычно проверяются на достоверность по заранее составленному списку возможных команд, и если команда окажется недостоверной, появится сообщение об ошибке.

В отличие от настольных прикладных программ, при написании таких веб-программ, как Joomla, приходится решать две непростые задачи. Прежде всего, веб-сайт открывается для всего сообщества пользователей Интернета, а среди них, к сожалению, имеются отдельные злонамеренные личности, пытающиеся раскрыть пароль администратора, испортить содержимое веб-сайта, заменив, например, некоторые его файлы своими собственными, или вообще вывести сайт из строя, разрушив информацию в его базе данных. Все это требует принятия практических мер защиты от подобных несанкционированных вмешательств в нормальную работу веб-сайта.

Еще одна трудная задача состоит в том, что количество команд, входящих в запрос, нельзя ни контролировать, ни ограничить. Как правило, команда представляет собой сочетание URL с рядом возможных значений полей из HTML-формы. Пользователи зачастую вводят команды, просто щелкая кнопкой мыши на ссылке или кнопке отправки формы, а следовательно, они всегда выдают действительные команды.

Но вполне возможно, что пользователь совершенно осознанно попытается ввести команду, чтобы сделать что-нибудь необычное или недозволенное, например, набрать URL вручную или изменить HTML-форму в окне браузера. К сожалению, веб-сервер не может ничего сообщить, выбрал ли пользователь ссылку или набрал URL вручную. И поэтому нельзя каким-то универсальным способом определить, заполнил ли пользователь форму и нажал кнопку ее предъявления или же видоизменил форму, чтобы передать какие-то данные со злым умыслом.

Ради перестраховки приходится всегда допускать, что команды, поступающие в запросе, могут быть специально предназначены для совершения атаки на веб-сайт, а следовательно, их нужно непременно проверить на достоверность, прежде чем выполнить.

Мы еще вернемся к вопросам безопасности и защиты в веб-программировании далее в книге, но они настолько важны, что заслуживают иллюстрации на конкретном примере уже теперь.

Допустим, имеется простая система, в которой любой посетитель веб-сайта может вводить комментарии к статьям, хотя утверждать их разрешается только зарегистрированным пользователям сайта. Комментарии не появятся на веб-сайте до тех пор, пока не будут утверждены. Следовательно, требуется организовать какую-то элементарную защиту против появления на веб-сайте неуместных комментариев.

Для целей данного примера выберем два поля: одно — для отображения комментариев, а другое — для сообщения об их утверждении или неутверждении. Реализовать это можно двумя способами. Когда отображается форма, мы должны проверить, зарегистрирован ли пользователь. Если он зарегистрирован, форма отображается в том виде, в каком она представлена на рис. 3.7.

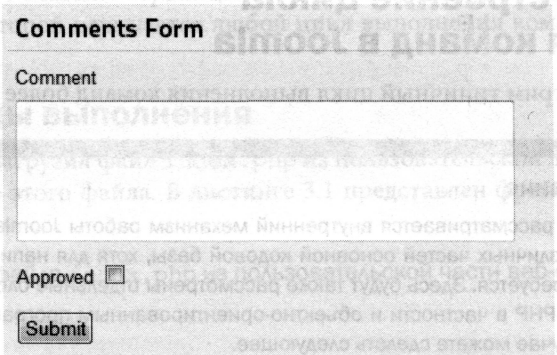


Рис. 3.7. Пример формы ввода комментариев к статьям

Итак, перед отображением формы мы должны проверить, зарегистрирован ли текущий пользователь, т.е. имеет ли он полномочия на утверждение комментариев к статьям. Если он не зарегистрирован, поле `Approved` (Утверждено) пропускается, а в форме отображается только поле `Comment` (Комментарий). Таким образом, незарегистрированные пользователи вообще не увидят поле `Approved`, а следовательно, не смогут установить одноименный флажок в форме.

На первый взгляд, такое проектное решение устраняет всякую возможность утверждения комментариев незарегистрированными пользователями, но на самом деле это не совсем так. Любой, разбирающийся в веб-программировании, может очень легко отредактировать HTML-разметку страницы в такой программе, как Firebug или Web Developer, чтобы отобразить на ней поле `Approved`, установив в нем флажок в состояние "утверждено". Предъявленная таким образом форма окажется утвержденной якобы зарегистрированным пользователем. Веб-серверу ничего не известно, была ли форма изменена до нажатия кнопки ее предъявления. Он просто обнаруживает данные этой формы, полученные среди прочих сведений в запросе.

Таким образом, данное проектное решение имеет серьезную брешь в защите. Как же ее устранить? Для этого можно, в частности, организовать дополнительную проверку перед обновлением базы данных. Несмотря на то что незарегистрированный пользователь не

может предъявить форму с флажком, установленным в поле Approved, мы, тем не менее, должны проверить этот факт еще раз, прежде чем записывать комментарии в базе данных. С этой целью мы должны убедиться в том, что пользователь зарегистрирован. И если он не зарегистрирован, мы можем всегда сбросить флажок в поле Approved и только после этого сохранить данные формы в базе данных. Благодаря этому недействительные данные не будут сохранены в базе данных, даже если незарегистрированный пользователь незаконно добавит поле Approved в предъявленную им форму, а следовательно, он не сможет нанести никакого вреда ни веб-сайту, ни тем, кто будет читать публикуемые на нем статьи.

Другие примеры нарушения защиты и безопасности будут представлены далее в этой книге по ходу изложения ее материала. Помимо этого, будут рассмотрены способы устранения подобных нарушений.

Внутреннее строение цикла выполнения команд в Joomla

А теперь рассмотрим типичный цикл выполнения команд более подробно.

Важное примечание

В этом разделе рассматривается внутренний механизм работы Joomla, чтобы стало понятнее назначение различных частей основной кодовой базы, хотя для написания программ для Joomla этого и не требуется. Здесь будут также рассмотрены отдельные блоки кода. Если вы незнакомы с языком PHP в частности и объектно-ориентированным программированием (ООП) вообще, в таком случае можете сделать следующее.

- Продолжить чтение, не обращая никакого внимания на особенности написания кода. Отдельные вопросы ООП и программирования на PHP будут рассматриваться в последующих главах книги по мере необходимости.
- Обращаться к материалу приложения А с кратким курсом ООП и PHP по мере рассмотрения кодовых блоков, чтобы лучше понять те действия, которые выполняются в каждой строке кода.
- Пропустить этот раздел и вернуться к нему после самостоятельного изучения особенностей программирования на PHP.

Загрузка файла index.php

Итак, начнем сеанс работы с веб-сайтом Joomla, набрав URL, указывающий на статью, опубликованную на главной странице веб-сайта, т.е. веб-адрес, аналогичный следующему: http://мой_сайт_joomla.org/index.php/название_моей_статьи.

Первое, что должно при этом произойти, зависит не от Joomla, а от типа веб-сервера (Apache или IIS). Веб-серверу ничего не известно о том, что мы указываем адрес динамического веб-сайта Joomla, и поэтому он попытается сначала найти файл или папку название_моей_статьи в папке index.php. Если ему не удастся это сделать, он перейдет к следующей части URL и попытается найти файл index.php. А поскольку этот файл существует, то он и выполняется. Аналогичным образом будут обрабатываться все URL в

пользовательской части веб-сайта, и поэтому каждый цикл выполнения команд в пользовательской части будет начинаться с загрузки веб-сервером файла `index.php`.

Если веб-сайт настроен на применение модуля `mod_rewrite` из веб-сервера Apache, мы можем и не указывать в URL файл `index.php` — он все равно будет загружен и выполнен. Следовательно, при указании любого URL, начинающегося с базового (в данном случае `http://мой_сайт_joomla.org`), веб-сервер Apache по-прежнему будет выполнять файл `index.php` в пользовательской части веб-сайта.

Это же справедливо и для всех URL административной части веб-сайта, за исключением того, что они указывают на файл `index.php` в папке `administrator`. По любому URL, начинающемуся с `http://myjoomlasite.org/administrator`, будет загружен и выполнен файл `index.php`, поскольку это будет первый файл, найденный веб-сервером для выполнения.

Таким образом, веб-сервер загрузит и выполнит файл `index.php` по любому URL, введенному, начиная с главной страницы веб-сайта. И этот файл служит типичной отправной точкой, с которой начинается любой цикл выполнения команд в пользовательской части веб-сайта.

Проверка среды выполнения

Итак, веб-сервер загрузил файл `index.php` из пользовательской части веб-сайта. Рассмотрим содержимое этого файла. В листинге 3.1 представлен фрагмент кода из первой части этого файла.

Листинг 3.1. Начало файла `index.php` из пользовательской части веб-сайта

```
<?php
/**
 * @package         Joomla.Site
 * @copyright       Copyright (C) 2005 - 2012 Open Source Matters, Inc. All
↳rights reserved.
 * @license         GNU General Public License version 2 or later; see
↳LICENSE.txt
 */
  // Установить признак того, что это родительский файл.
  define('_JEXEC', 1);
```

Первая часть данного файла содержит комментарии и называется блоком документации (doc). Строки кода с этими комментариями присутствуют (и должны *непременно* быть) в каждом файле базового кода PHP, применяемом в Joomla. Дескриптор `@package` обозначает имя файла пакета. Для файлов из пользовательской части веб-сайта обычно указывается имя `Joomla.Site`, а для файлов из административной части — имя `Joomla.Administrator`. В строках кода с дескрипторами `@copyright` и `@license` указываются одни и те же для каждого файла сведения об авторском праве и лицензионном соглашении.



Далее следует строка кода `define('_JEXEC', 1);`, имеющая очень большое значение. В ней определяется константа `'_JEXEC'`. В начале практически всех файлов исходного кода PHP, за исключением файла `index.php` из административной части, содержится строка кода `defined('_JEXEC', 1) or die;`. В этой строке проверяется, определена ли константа `'_JEXEC'`, т.е. осуществляется ли вход в приложение из файла `index.php`.

Если кто-нибудь попытается выполнить файл исходного кода PHP непосредственно, т.е. вне приложения, константа `'_JEXEC'` окажется неопределенной, и файл не будет выполнен. Если же включить в состав приложения файл исходного кода PHP, в начале которого отсутствует упомянутая выше строка кода с определением данной константы, то кто-нибудь может попытаться ввести путь к этому файлу непосредственно в браузере. Как правило, если файл исходного кода PHP для Joomla выполняется за пределами системы CMS, появляется сообщение об ошибке в коде PHP (например, из-за того, что не определена константа или класс). В этом сообщении указывается полный путь к файлу исходного кода PHP. С точки зрения безопасности это представляет определенный риск, поскольку потенциальный нарушитель получает сведения о сервере и находящихся на нем папках.

Убедитесь в этом сами, удалив строку кода `defined('_JEXEC', 1) or die;` из файла `components/com_banners/banners.php`. Затем введите в окне браузера следующий URL: `http:<ваш_сайт_joomla>/components/com_banners/banners.php`. В итоге вы должны получить сообщение об ошибке, аналогичное приведенному на рис. 3.8.

Fatal error: Call to undefined function jimport() in C:\xampp\htdocs\netbeans_7\joomla\components\com_banners\banners.php on line *13*

Call Stack

#	Time	Memory	Function	Location
1	0.0015	340120	{main}()	..\banners.php:0

Рис. 3.8. Пример раскрытия полного пути к файлу

Как видите, в сообщении об ошибке указан полный путь к установленным программным компонентам Joomla. Если вы восстановите удаленную строку кода с определением упомянутой выше константы и сделаете еще одну попытку, то получите в итоге пустой экран, не обнаруживающий на этот раз никаких сведений о веб-сайте.

Определение мест расположения файлов

Фрагмент кода из следующего раздела файла `index.php` представлен в листинге 3.2. В нем определяется целый ряд констант, задающих места расположения файлов.

Листинг 3.2. Загрузка файла `defines.php`

```
define('DS', DIRECTORY_SEPARATOR);

if (file_exists(dirname(__FILE__) . '/defines.php')) {
  include_once dirname(__FILE__) . '/defines.php';
}

if (!defined('_JDEFINES')) {
  define('JPATH_BASE', dirname(__FILE__));
  include_once JPATH_BASE.DS.'includes'.DS.'defines.php';
}
```

Сначала определяется константа DS. Эта константа устарела, поскольку теперь она больше не нужна. Она определяет символ разделения каталогов, которым в Linux и Mac OS X является *прямая* косая черта (/), а в Windows — *обратная* косая черта (\). Но поскольку прямая косая черта теперь правильно интерпретируется в версии PHP для Windows, то от константы DS можно благополучно избавиться.

В двух последующих кодовых блоках с условным оператором `if` предоставляются два возможных способа определения ряда констант. В состав Joomla входит стандартный файл `includes/defines.php`, поэтому в первом блоке с условным оператором `if` поиск файла `defines.php` осуществляется в папке верхнего уровня. В стандартной установке такой файл отсутствует, и поэтому первый блок с условным оператором `if` не выполняется. А поскольку константа `'JPATH_BASE'` еще не определена, то стандартный файл `includes/defines.php` выполняется по команде `include_once` языка PHP.

В файле `includes/defines.php` определяются константы, приведенные в листинге 3.3.

Листинг 3.3. Стандартный файл `defines.php`

```
// Глобальные определения.
// Определения путей к интегрированной среде Joomla!.
$parts = explode(DS, JPATH_BASE);

// Определения констант.
define('JPATH_ROOT', implode(DS, $parts));

define('JPATH_SITE', JPATH_ROOT);
define('JPATH_CONFIGURATION', JPATH_ROOT);
define('JPATH_ADMINISTRATOR', JPATH_ROOT.DS.'administrator');
define('JPATH_LIBRARIES', JPATH_ROOT.DS.'libraries');
define('JPATH_PLUGINS', JPATH_ROOT.DS.'plugins' );
define('JPATH_INSTALLATION', JPATH_ROOT.DS.'installation');
define('JPATH_THEMES', JPATH_BASE.DS.'templates');
define('JPATH_CACHE', JPATH_BASE.DS.'cache');
define('JPATH_MANIFESTS', JPATH_ADMINISTRATOR.DS.'manifests');
```

Каждая из приведенных выше констант служит для обнаружения папки, в которой находятся различные типы основных файлов Joomla. Если стандартный файл `defines.php` требуется заменить, для этого достаточно создать файл с именем `defines.php` в папке верхнего уровня и определить в нем требующиеся константы. Если при выполнении этого файла определяется константа `JPATH_BASE`, то стандартный файл `includes/defines.php` не будет выполняться.

Это первый пример простого способа внесения корректив в стандартную установку Joomla. Так, если требуется переместить файлы в другие папки исходной установки Joomla, для этого достаточно создать собственный специальный файл `defines.php` и указать пути к новым папкам.

С помощью констант PHP можно один раз определить местоположение каждого файла для всего приложения. Когда же потребуется найти эти файлы в других частях приложения, достаточно обратиться к соответствующим константам в полной уверенности, что они правильно укажут местоположение файлов в текущей установке Joomla.

Диалог авторов по поводу включения файлов исходного кода PHP

Марк: Луис, мне непонятно, каким образом исполняются или включаются файлы PHP. Ведь мы начинаем с одного файла `index.php`, а затем выполняем целый ряд других файлов. Какой в этом смысл?

Луис: Возможность включать другие файлы на самом деле является весьма примечательной особенностью PHP. Ведь она позволяет нам поддерживать относительно малые размеры файлов, уделяя основное внимание одной решаемой задаче, а затем объединять их вместе по мере надобности. Конечно, мы могли бы разместить все приложение Joomla в одном очень крупном файле сценария, но насколько просто было бы с ним обращаться?

Марк: Я, конечно, догадываюсь, что с мелкими файлами, даже если их много, гораздо проще разобраться и сопровождать. Но мне все-таки непонятно, как это делается. Если посмотреть исходный код, то в нем можно обнаружить по меньшей мере четыре разные команды для включения файлов: `include_once`, `require_once`, `jimport` и `require`. Не выполняют ли все они, по существу, одно и то же: файл исходного кода PHP?

Луис: И да, и нет. Все они выполняют файл исходного кода PHP под определенным именем, но делают это по-разному.

Например, команда `include_once` не остановит выполнение программы, если файл не будет найден, тогда как команда `require_once` остановит. А команда `jimport` используется в том случае, если во включаемых файлах применяются условные обозначения библиотеки Joomla. Эта команда выполняет вместе с тем ряд других полезных для Joomla функций, в том числе проверку имен вспомогательных классов. Поэтому для использования разных команд обычно имеется немало веских оснований.

Команды `include_once` и `require_once` не загружают один и тот же файл дважды. И это очень важно, если файл содержит объявление класса и, возможно, уже включен. Если попытаться объявить класс еще раз, то будет выдано сообщено об ошибке в коде PHP. С другой стороны, используя эти команды в коде PHP, приходится проверять, был ли файл уже включен. В некоторых системах, особенно на крупных серверах в сетях хранения данных (SAN), это может стать причиной снижения производительности Joomla.

Марк: Какой же командой мне все-таки пользоваться, когда я разрабатываю свои собственные расширения?

Луис: Это очень важный вопрос. На самом деле мы вообще не должны пользоваться командами `include_once` и `require_once` в Joomla, поскольку они замедляют работу веб-сайта на некоторых платформах. Оказывается, что намного быстрее просто отслеживать классы, загружаемые в приложение, вместо того, чтобы поручать коду PHP следить за теми файлами, которые были загружены.

Включая другие файлы, мы зачастую загружаем классы. Так, для включения файлов типа `class` имеется отличный метод `JLoader::register()` из файла `libraries/loader.php`. Замечательная особенность этого метода состоит в том, что он использует средство автоматической загрузки PHP, чтобы загружать классы в оперативную память только по мере надобности. Этот метод просто добавляет файл типа `class` в список файлов, которые могут быть автоматически загружены при вызове из кода PHP. Но если этот файл не нужен, то он не загружается в оперативную память. Этот метод загрузки файлов с классами действует намного быстрее, чем команда `require_once`, — по крайней мере, на некоторых аппаратных платформах.

Марк: Это действительно важно. Но ведь многие файлы исходного кода PHP для Joomla на самом деле содержат только объявления классов и ничего фактически не делают при выполнении команды `jimport`, `require_once` или `include_once`. Как быть со всем этим?

Луис: Как известно, на PHP можно составлять процедурные сценарии или файлы, создающие классы. Если же файл содержит только объявление класса, т.е. объявляет класс с полями и методами, то при его включении никакой код, по существу, не выполняется. Код выполняется лишь при создании объекта данного класса или одного из его методов.

В этом как раз и проявляются замечательные особенности метода `JLoader::register()`. Ведь он не загружает файл с классом в оперативную память до тех пор, пока не потребуется выполнить один из методов этого класса. При желании можно было бы даже автоматически регистрировать каждый класс в Joomla, используя этот метод в ходе процедуры инициализации, после чего основные классы стали бы сразу же доступными без дополнительных хлопот. Единственная причина не делать этого — дополнительное расходование оперативной памяти на хранение всех имен классов и файлов.

Марк: Пожалуй, в этом есть определенный смысл. А как насчет средства автоматической загрузки, внедренного в версии Joomla 2.5? И что вообще представляет собой автоматический загрузчик?

Луис: Автоматический загрузчик — это очередной шаг на пути к облегчению труда разработчиков в Joomla. Он действует по очень простому принципу. Если попытаться использовать стандартный класс платформы Joomla или ее системы CMS, еще не загруженный в оперативную память, автоматический загрузчик сам загрузит его, избавляя разработчика от необходимости обращаться к команде `jimport` или классу `JLoader`. Поэтому для использования большинства классов платформы разработчику вообще не нужно иметь дело с командами включения или загрузки.

Загрузка интегрированной среды Joomla

В следующей строке кода интегрированная среда Joomla загружается из файла `includes/framework.php`:

```
require_once JPATH_BASE.DS.'includes'.DS.'framework.php';
```

Файл `framework.php` выполняет ряд очень важных функций. Прежде всего он загружает файл `libraries/joomla/import.php`, который, в свою очередь, загружает ряд важных классов, в том числе и класс `JFactory` из файла `libraries/joomla/factory.php`. В ходе этого процесса проверяется наличие файла `configuration.php`. Если этот файл отсутствует, придется обратиться к приложению `installation`, чтобы попробовать установить Joomla. А если установочные файлы не удастся найти, будет выдано соответствующее сообщение об ошибке, и данный процесс завершится. И наконец, этот файл импортирует классы интегрированной среды. Эта операция выполняется в строках кода, приведенных в листинге 3.4.

Листинг 3.4. Импорт классов из интегрированной среды Joomla

```
//
// Импорт классов из библиотеки Joomla!.
//

jimport('joomla.application.menu');
jimport('joomla.environment.uri');
jimport('joomla.utilities.utility');
jimport('joomla.event.dispatcher');
jimport('joomla.utilities.arrayhelper');
```

Следует, однако, иметь в виду, что файл `libraries/joomla/import` был уже выполнен раньше из файла `framework.php`. В сценарии из этого файла был загружен класс `JLoader`, который, в свою очередь, предоставляет метод `jimport()`. Это дает возможность воспользоваться методом `jimport()` для импорта остальных классов интегрированной среды Joomla. Как упоминалось раньше при рассмотрении циклов выполнения команд, каждый такой цикл должен быть автономным. Именно поэтому все эти классы приходится загружать в оперативную память в каждом цикле выполнения команд.

Итак, мы рассмотрели уже немало, хотя все еще находимся в самом начале сеанса работы с веб-сайтом Joomla. По-настоящему Joomla начинает действовать лишь после загрузки интегрированной среды.

Начало или продолжение сеанса

Итак, вернемся к файлу `index.php` и рассмотрим в нем следующий ряд строк кода, приведенных в листинге 3.5.

Листинг 3.5. Методы `getApplication()` и `initialise()` из файла `index.php`

```
//  пометить состояние после загрузки в профилировщике
JDEBUG ? $_PROFILER->mark('afterLoad') : null;

// получить экземпляр приложения
$app = JFactory::getApplication('site');

// инициализировать приложение
$app->initialise();

// пометить состояние после инициализации в профилировщике
JDEBUG ? $_PROFILER->mark('afterInitialise') : null;
```

Не будем пока что обращать особого внимания на два оператора отладки `JDEBUG` в этом фрагменте кода. Ведь при нормальной работе они пропускаются. Эти операторы помогают точнее оценить производительность отдельных частей кода.

Итак, в данном фрагменте кода сначала создается объект приложения, сохраняемый в переменной `$app`, а затем выполняется его метод `initialise()`. В результате создания объекта приложения загружается класс `JSite` из файла `includes/application.php`. Этот класс служит расширением класса `JApplication`. Аналогичный класс `JAdministrator` из файла `administrator/includes/application.php` служит для создания объекта приложения в административной части веб-сайта.

С этого момента, собственно, и начинается сам сеанс работы с веб-сайтом Joomla. Мы загружаем уже имеющийся сеанс или создаем новый, если ничего другого не найдено. Напомним, что Joomla ничего не известно о том, что происходило в предыдущих циклах выполнения команд, за исключением того, что было сохранено в сеансе работы или базе данных. Именно поэтому так важно найти существующий сеанс, чтобы перенести информацию из предыдущего цикла в следующий. Кроме того, в данный момент загружается файл `configuration.php`, предоставляющий доступ к базе данных Joomla и другим настройкам данного конкретного веб-сайта.

В данный момент выполняется еще одна важная операция: создание поля `requestTime`. Это поле из класса `JApplication` содержит отметку даты и времени по Гринвичу,

с которых начинается цикл выполнения запросов. Если же требуется поле для отметки текущих даты и времени, этого нетрудно добиться с помощью вызова метода `$app->requestTime()` или `JFactory::getApplication()->requestTime()`. В отличие от результата вызова метода `now()`, значение подобной отметки остается постоянным в течение всего цикла выполнения команд.

Далее выполняется метод `initialise()`, инициализирующий приложение. А поскольку мы находимся в классе `JSite`, то данный метод выполняется из файла `includes/application.php`. Основное назначение этого метода — сначала выяснить, какой именно язык требуется пользователю, а затем загрузить объект этого языка, чтобы правильно перевести текстовые строки в пользовательской части веб-сайта. А в самом конце этого метода вызывается также метод `initialise()` из родительского класса (`parent::initialise($options);`). Это приводит к вызову метода `initialise()` класса `JApplication` из файла `libraries/joomla/application/application.php`, чтобы выяснить, какой именно редактор подходит данному пользователю.

Маршрутизация URL

В следующем разделе файла `index.php` находится фрагмент кода, приведенный в листинге 3.6.

Листинг 3.6. Метод `route()` маршрутизации приложения

```
// Маршрутизация приложения.
$app->route();

// пометить состояние после маршрутизации приложения в профилировщике
JDEBUG ? $_PROFILER->mark('afterRoute') : null;
```

В терминологии Joomla *маршрутизатором* называется класс, преобразующий URL в массив команд. В классе маршрутизатора имеются два открытых метода: `parse()` и `build()`. Метод `parse()` принимает объект типа `JURI` в качестве входного параметра и возвращает массив команд. А метод `build()` выполняет обратный процесс, принимая массив команд в качестве своего параметра и возвращая объект типа `JURI`, преобразованный из данного массива. Рассмотрим особенности этого преобразования более подробно. Напомним, что в переменной `$app` хранится объект типа `JSite`. В листинге 3.7 показано, что же происходит в методе `route()` класса `JSite` из файла `includes/application.app`.

Листинг 3.7. Метод `route()` из класса `JSite`

```
public function route()
{
  parent::route();

  $Itemid = JRequest::getInt('Itemid');
  $this->authorise($Itemid);
}
```

Напомним, что класс `JSite` служит расширением класса `JApplication` из файла `libraries/joomla/application/application.php`. В классе `JApplication` также имеется метод `route()`. Поэтому метод `route()` переопределяет в классе `JSite`

одноименный метод из класса JApplication. Прежде всего он вызывает метод route() из родительского класса (JApplication) по команде parent::route(). Исходный код метода route() из родительского класса JApplication приведен в листинге 3.8.

Листинг 3.8. Метод route() из класса JApplication

```
public function route()
{
  // получить полный запрос URI
  $uri = clone JURI::getInstance();

  $router = $this->getRouter();
  $result = $router->parse($uri);

  JRequest::set($result, 'get', false);

  // инициировать событие onAfterRoute
  JPluginHelper::importPlugin('system');
  $this->triggerEvent('onAfterRoute');
}
```

В коде этого метода получается текущий объект типа JURI. Команда clone предписывает системе сделать копию объекта типа JURI вместо того, чтобы использовать исходный объект. Это делается для того, чтобы мы неумышленно не видоизменили текущий URI (Универсальный идентификатор ресурса) при обращении с объектом типа JURI. Сделав копию этого объекта, мы можем далее работать именно с ней, например, видоизменить ее, не меняя оригинал.

В итоге мы получаем URL для последующего синтаксического анализа. В следующей строке кода получается конкретный объект класса маршрутизатора. В данном случае мы находимся в пользовательской части веб-сайта, иначе называемой сайтом, и поэтому при выполнении команды $this->getRouter() получаем объект типа JRouterSite (одноименного класса из файла includes/router.php).

Далее выполняется метод parse() маршрутизатора, возвращающий ассоциативный массив команд. В рассматриваемом здесь примере объект типа JURI будет содержать сведения об URL для пункта меню выбора статьи (http://мой_сайт_joomla.org/index.php/название_моей_статьи), а в массив команд он превращается следующим образом:

- Itemid — **123** (идентификатор пункта меню)
- option — **com_content** (наименование компонента)
- view — статья (наименование представления)
- id — **234** (идентификатор статьи)

Вся эта информация сохраняется обратно в переменной $_GET языка PHP с помощью метода JRequest::set(). И наконец, инициируется системное событие onAfterRoute для выполнения любых системных подключаемых модулей при наступлении этого события. Подробнее о событиях речь пойдет в главе 5.

На этом выполнение метода route() из родительского класса завершается и происходит возврат к методу route() из класса JSite. В итоге мы получаем Itemid — столбец идентификаторов из базы данных для пункта меню, который нам требуется отобразить.

Вызывая далее метод `$this->authorise()`, мы проверяем, уполномочен ли текущий пользователь просматривать этот пункт меню. И наконец, мы возвращаем файл `index.php` и выполняем команду `DEBUG`.

Выполнение компонента

Теперь нам известно, что делать дальше. Введенный пользователем URL (например, щелчком на ссылке на пункт меню) проверен, преобразован в массив команд и сохранен в переменной `$_GET` языка PHP. Итак, мы готовы выполнить компонент.

Возвращаясь к файлу `index.php`, мы обнаруживаем следующую часть исходного кода, приведенного в листинге 3.9.

Листинг 3.9. Вызов метода `dispatch()`

```
// произвести диспетчеризацию приложения
$app->dispatch();

// пометить состояние после диспетчеризации в профилировщике
JDEBUG ? $_PROFILER->mark('afterDispatch') : null;
```

Метод `dispatch()` фактически выполняет компонент. Рассмотрим, каким образом это делается. В листинге 3.10 приведен исходный код метода `dispatch()`.

Листинг 3.10. Исходный код метода `dispatch()` из класса `JSite`

```
public function dispatch($component = null)
{
  try
  {
    // получить компонент, если он не задан
    if (!$component) {
      $component = JRequest::getCmd('option');
    }

    $document = JFactory::getDocument();
    $user = JFactory::getUser();
    $router = $this->getRouter();
    $params = $this->getParams();

    switch($document->getType())
    {
      case 'html':
        // получить язык
        $lang_code = JFactory::getLanguage()->getTag();
        $languages = JLanguageHelper::getLanguages('lang_code');

        // задать метаданные
        if (isset($languages[$lang_code]) && $languages[$lang_code]->metakey)
{
          $document->setMetaData('keywords', $languages[$lang_code]->metakey);
        } else {
           $document->setMetaData('keywords', $this->getCfg('MetaKeys'));
        }
```

```
        $document->setMetaData('rights', $this->getCfg('MetaRights'));
        if ($router->getMode() == JROUTER_MODE_SEF) {
          $document->setBase(htmlspecialchars(JURI::current()));
        }
        break;

      case 'feed':
        $document->setBase(htmlspecialchars(JURI::current()));
        break;
    }

    $document->setTitle($params->get('page_title'));
    $document->setDescription($params->get('page_description'));
    $contents = JComponentHelper::renderComponent($component);
    $document->setBuffer($contents, 'component');

    // Инициировать событие onAfterDispatch.
    JPluginHelper::importPlugin('system');
    $this->triggerEvent('onAfterDispatch');
  }
  // обработать любые неперехваченные исключения
  catch (Exception $e)
  {
    $code = $e->getCode();
    JError::raiseError($code ? $code : 500, $e->getMessage());
  }
}
```

Прежде всего следует обратить внимание на то, что весь код данного метода выполняется в блоке `try/catch`. Это означает, что код выполняется в следующей последовательности.

- Код в фигурных скобках после оператора `try` (т.е. в блоке `try`) выполняется как обычно.
- Если выполнение любого оператора в блоке `try` приводит к ошибке, управление программой передается первой строке кода в блоке `catch`, находящемся в самом конце данного метода.

Более подробно блоки `try/catch` будут рассмотрены в главе 8. В блоке `try` имя компонента получается по команде `option`, выявленной ранее в результате синтаксического анализа URL. Затем создается новый документ в виде объекта `$document`. Этот объект будет использоваться далее для хранения всей информации о странице, которая в конечном итоге отправляется обратно браузеру.

В отношении документа необходимо иметь в виду следующее. Во-первых, это объект, находящийся в оперативной памяти, и поэтому нам не обязательно создавать его последовательно. Мы можем вставлять отдельные части документа в любом удобном для нас порядке. Выгодно используя эту особенность, мы можем составить документ в непоследовательном порядке, например, добавив сначала компонент, а затем вставив модули.

И во-вторых, в Joomla поддерживаются разные типы документов. При просмотре сайта мы обычно имеем дело с документом типа `html`. Документ данного типа требует

некоторой специальной обработки. В частности, мы получаем язык для отображения текста на странице, задаем метаданные и базу документа, если используются URL типа SEF (т.е. удобные для поиска URL). Так, если документ представляет собой ленту новостей (`type = "feed"`), мы пропускаем большую часть процесса его обработки и только задаем базу документа. А если документ относится к любому другому типу, мы вообще не выполняем его специальную обработку, пропуская ее полностью.

После установки заголовка и описания страницы мы наконец доходим до самого главного, как показано в приведенной ниже строке кода.

```
$contents = JComponentHelper::renderComponent($component);
```

В этой строке кода выполняется заданный нами компонент, а результат сохраняется в переменной `$contents`. Рассмотрим подробнее, как это делается. В данной строке кода вызывается статический метод `renderComponent()` класса `JComponentHelper` из файла `libraries/joomla/application/component/helper.php`. Первая часть исходного кода этого метода приведена в листинге 3.11.

Листинг 3.11. Метод `renderComponent()` из класса `JComponentHelper`. Часть I

```
public static function renderComponent($option, $params = array())
{
  // инициализировать переменные
  $app = JFactory::getApplication();

  // загрузить файлы шаблонов языковой поддержки
  $template = $app->getTemplate(true)->template;
  $lang = JFactory::getLanguage();
    $lang->load('tpl_'.$template, JPATH_BASE, null, false, false)
  || $lang->load('tpl_'.$template, JPATH_THEMES."/$template", null,
↳false, false)
  || $lang->load('tpl_'.$template, JPATH_BASE, $lang->getDefault(),
↳false, false)
  || $lang->load('tpl_'.$template, JPATH_THEMES."/$template",
↳$lang->getDefault(), false, false);

  if (empty($option)) {
    // сгенерировать сообщение об ошибке 404, если компонент отсутствует
    JError::raiseError(404,
↳JText::_('JLIB_APPLICATION_ERROR_COMPONENT_NOT_FOUND'));
    return;
  }

  // записать область действия
  $scope = $app->scope;
  // задать имя компонента в области действия
  $app->scope = $option;

  // сформировать путь к компоненту
  $option = preg_replace('/[^A- Z0- 9_\.- ]/i', '', $option);
  $file = substr($option, 4);

  // определить путь к компоненту
```

```
  define('JPATH_COMPONENT', JPATH_BASE . '/components/' . $option);
  define('JPATH_COMPONENT_SITE', JPATH_SITE . '/components/' . $option);
  define('JPATH_COMPONENT_ADMINISTRATOR', JPATH_ADMINISTRATOR .
↳'/components/' . $option);

  // получить путь к компоненту
  if ($app->isAdmin() && file_exists(JPATH_COMPONENT . '/admin.'.$file.'.php'))
{
    $path = JPATH_COMPONENT . '/admin.'.$file.'.php';
  } else {
    $path = JPATH_COMPONENT . '/' . $file.'.php';
  }

  // сгенерировать сообщение об ошибке, если компонент недоступен
  if (!self::isEnabled($option) || !file_exists($path)) {
    JError::raiseError(404,
↳JText::_('JLIB_APPLICATION_ERROR_COMPONENT_NOT_FOUND'));
  }

  $task = JRequest::getString('task');
  // загрузить общие и локальные файлы языковой поддержки
    $lang->load($option, JPATH_BASE, null, false, false)
  || $lang->load($option, JPATH_COMPONENT, null, false, false)
  || $lang->load($option, JPATH_BASE, $lang->getDefault(), false, false)
  || $lang->load($option, JPATH_COMPONENT, $lang->getDefault(), false,  false);
```

В приведенной выше первой части метода `renderComponent()` загружается шаблон и его язык. Затем имя компонента сохраняется в поле `scope` объекта типа `JSite` (`$app->scope = $option`). Это очень удобное поле, поскольку в любой момент цикла выполнения команд мы можем проверить содержимое этого поля, чтобы выяснить, какой именно компонент выполняется. Например, мы можем воспользоваться этим обстоятельством, чтобы выполнить подключаемый модуль только для выбранных компонентов.

В строке кода с командой `preg_replace` проверяется наличие в переменной `$option` только букв, чисел и знаков подчеркивания. А все остальные символы заменяются знаками тире. Благодаря этому исключаются любые символы, способные так или иначе исказить имя файла. В следующей строке кода выполняется буквально следующее: использовать часть имени в переменной `$option` после четвертой позиции в качестве имени файла. Это условное обозначение имен компонентов. В рассматриваемом здесь примере компонент называется `com_content`, и поэтому файл, который нужно выполнить, называется `content.php`. Более подробно эти и другие условные обозначения имен будут рассмотрены в следующем разделе.

Итак, мы практически готовы выполнить заданный нами компонент! В следующем фрагменте кода определяется путь к файлам компонента в пользовательской и административной частях веб-сайта, а переменная `$path` устанавливается в зависимости от принадлежности данного компонента к пользовательской или административной части веб-сайта. После этого мы должны убедиться в том, что компонент активизирован. И наконец, мы вновь прибегаем к логической операции ИЛИ для загрузки файлов языковой поддержки из одного из возможных мест их расположения.

Предупреждение о наличии сложного кода!

В начале и конце кода из листинга 3.11 имеются четыре строки с операторами `$lang->load`, объединенными вместе логической операцией ИЛИ (`||`). Эта логическая операция часто используется вместо условного оператора `if`, когда требуется проверить истинность нескольких условий. Но логическая операция ИЛИ обладает еще одной интересной особенностью: она останавливает обработку, как только доходит до выражения, логическое значение которого вычисляется как "истина" (`true`).

Метод `load()` из класса `JLanguage` возвращает логическое значение `true`, если загрузка произведена успешно, а иначе — логическое значение `false`. Следовательно, используя логическую операции ИЛИ, можно быстро (хотя и, на первый взгляд, сложно) выполнить следующие действия: попытаться загрузить файл языковой поддержки, начиная с места, определяемого сначала константой `JPATH_BASE`, затем константой `JPATH_THEMES` и т.д., а между тем — остановить процесс загрузки, как только один из файлов будет загружен успешно.

В следующем фрагменте кода, приведенном в листинге 3.12, фактически загружается файл заданного нами компонента.

Листинг 3.12. Метод `renderComponent()` из класса `JComponentHelper`. Часть II

```
// наметить область предварительного просмотра шаблона
$contents = null;

// выполнить компонент
$contents = self::executeComponent($path);
```

В этом фрагменте кода сначала очищается все, что может находиться в переменной `$contents`, а затем вызывается метод `executeComponent()`. Обратите внимание на применение ключевого слова `self` в вызове данного метода. Дело в том, что метод `executeComponent()` является статическим методом из текущего класса. Более подробно ключевое слово `self` рассматривается в приложении А. Ниже приведен код метода `executeComponent()`:

```
protected static function executeComponent($path)
{
  ob_start();
  require_once $path;
  $contents = ob_get_contents();
  ob_end_clean();
  return $contents;
}
```

Сначала функция `ob_start()` языка PHP предписывает активизировать буферизацию вывода. В итоге все, что выводится из компонента, будет сохраняться в оперативной памяти. Затем по команде `require_once $path` загружается файл компонента. В рассматриваемом здесь примере загружается файл `components/content.php`.

Более подробно компоненты и применяемый в Joomla шаблон проектирования MVC (Модель-Представление-Контроллер) рассматриваются в главах 7 и 10. А до тех пор важно понять, что для выполнения компонента в Joomla требуется лишь одно: загрузить

упомянутый выше файл, путь к которому указан в переменной $path, а все остальное зависит от самого компонента.

Это, в частности, означает, что компонент совсем не обязательно должен соответствовать шаблону MVC или даже использовать классы и объекты. Компонент может представлять собой отдельный сценарий, если он придерживается принятых условных обозначений имен. Впрочем, для создания компонентов по шаблону MVC имеется немало веских оснований, о чем будет сказано в соответствующем месте при построении компонентов.

Для того чтобы ощутить на практике, каким образом действует компонент, попробуйте создать собственный компонент com_test, следуя инструкциям в документации Joomla по адресу http://docs.joomla.org/Testing_Checklists#Testing_Code_Snippets. Как только вы введете ссылку ?option=com_test в URL, ваш пробный компонент будет выполнен из файла test.php, находящегося в папке components/com_test.

После того как завершится выполнение компонента, содержимое буфера сохраняется обратно в переменной $clean с помощью метода ob_get_contents(). И наконец, буфер очищается с помощью метода ob_end_clean().

Итак, компонент выполнен, а выводимый из него результат сохранен в переменной $contents. Осталось лишь рассмотреть последнюю часть метода renderComponent() из класса JComponentHelper. Соответствующий код приведен в листинге 3.13.

Листинг 3.13. Метод renderComponent() из класса JComponentHelper. Часть III

```
  // построить панель инструментов компонента
  jimport('joomla.application.helper');

  if (($path = JApplicationHelper::getPath('toolbar')) && $app->isAdmin()) {
    // получить задачу снова на тот случай, если она изменилась
    $task = JRequest::getString('task');

    // построить панель инструментов
    include_once $path;
  }

  // вернуться к прежней области действия
  $app->scope = $scope;

  return $contents;
}
```

В этой, последней части метода renderComponent() сначала проверяется, следует ли отображать панель инструментов, и если это нужно, то она отображается. Затем устанавливается прежнее значение переменной $app->scope. И в конце данного метода возвращается переменная $contents. Она передается вызывающему методу, которым в данном случае оказывается метод dispatch() из класса JSite.

В следующей строке кода из упомянутого выше метода dispatch() переменная $contents загружается в компонентную часть документа:

```
$document->setBuffer($contents, 'component');
```

И наконец, мы инициируем событие onAfterDispatch для системных подключаемых модулей. А если мы отлаживаем код, то помечаем момент наступления события

onAfterDispatch после диспетчеризации. Теперь результат выполнения компонента находится в документе, и все готово, чтобы разместить его в других местах страницы.

Воспроизведение страницы

Ниже приведен фрагмент кода из следующей части файла index.php.

```
// воспроизвести приложение
$app->render();
```

На данном этапе у нас имеется результат выполнения компонента, но отсутствует какое-либо форматирование и другое содержимое страницы, например, модули или сообщения. В листинге 3.14 приведен исходный код метода $app->render(), отвечающего за воспроизведение страницы.

Листинг 3.14. Метод render() из класса JSite

```
public function render()
{
  $document = JFactory::getDocument();
  $user = JFactory::getUser();

  // получить формат для воспроизведения страницы
  $format = $document->getType();

  switch ($format)
  {
    case 'feed':
      $params = array();
      break;

    case 'html':
    default:
      $template = $this->getTemplate(true);
      $file = JRequest::getCmd('tmpl', 'index');

      if (!$this->getCfg('offline') && ($file == 'offline')) {
        $file = 'index';
      }

      if ($this->getCfg('offline') && !$user->authorise('core.admin')) {
          $uri = JFactory::getURI();
          $return = (string)$uri;
          $this->setUserState('users.login.form.data',
↳array( 'return' => $return));
          $file = 'offline';
          JResponse::setHeader('Status', '503 Service Temporarily
↳Unavailable', 'true');
      }

      if (!is_dir(JPATH_THEMES.DS.$template->template)
↳&& !$this->getCfg('offline')) {
          $file = 'component';
          }
```

```
            $params = array(
                'template' => $template->template,
                'file' => $file.'.php',
                'directory' => JPATH_THEMES,
                'params' => $template->params
            );
            break;
        }

        // произвести синтаксический анализ документа
        $document = JFactory::getDocument();
        $document->parse($params);

        // инициировать событие onBeforeRender
        JPluginHelper::importPlugin('system');
        $this->triggerEvent('onBeforeRender');

        $caching = false;
        if ($this->getCfg('caching') && $this->getCfg('caching',2) == 2
↳&& !$user->get('id')) {
            $caching = true;
        }

        // воспроизвести документ
        JResponse::setBody($document->render($caching, $params));
        // инициировать событие onAfterRender
        $this->triggerEvent('onAfterRender');
    }
```

Код данного метода начинается с получения документа пользователя, а также типа или формата документа. В зависимости от конкретного формата в операторе `switch` выполняется соответствующая обработка. В рассматриваемом здесь примере выбирается документ типа `html`, поэтому мы получаем его шаблон и устанавливаем массив в переменной `$params`.

После оператора `switch` вызывается метод `parse()` для синтаксического анализа документа, как показано в листинге 3.15. А поскольку документ относится к типу `html`, то объект документа относится к классу `JDocumentHTML` из файла `libraries/joomla/document/html/html.php`.

Листинг 3.15. Метод `parse()` из класса `JDocumentHTML`

```
public function parse($params = array()) {
        return $this->_fetchTemplate($params)->_parseTemplate();
}
```

В первой части оператора возврата из данного метода загружается шаблон и соответствующий файл языковой поддержки. А поскольку метод `_fetchTemplate()` возвращает текущий объект, мы можем сначала связать методы в цепочку, а затем вызвать метод `_parseTemplate()`. Более подробно связывание методов в цепочку рассматривается в приложении А. Метод `_parseTemplate()`, исходный код которого приведен в листинге 3.16, проверяет файл `index.php` шаблона и создает массив со всеми элементами разметки `jdoc:include`.

Листинг 3.16. Метод `_parseTemplate()` из класса `JDocumentHTML`

```
protected function _parseTemplate()
{
  $matches = array();

  if (preg_match_all('#<jdoc:include\ type="([^"]+)" (.*)\/>#iU',
↳$this->_template, $matches))
  {
    $template_tags_first = array();
    $template_tags_last = array();

    // просмотреть элементы jdoc в обратном порядке
    for ($i = count($matches[0]) - 1; $i >= 0; $i--)
    {
      $type = $matches[1][$i];
      $attribs = empty($matches[2][$i]) ? array() :
↳JUtility::parseAttributes($matches[2][$i]);
      $name = isset($attribs['name']) ? $attribs['name'] : null;

      // разделить буферы на первый и последний по порядку выполнения
      if ($type == 'module' || $type == 'modules')
      {
        $template_tags_first[$matches[0][$i]] = array('type' => $type,
↳'name' => $name, 'attribs' => $attribs);
      }
      else
      {
        $template_tags_last[$matches[0][$i]] = array('type' => $type,
↳'name' => $name, 'attribs' => $attribs);
      }
    }

    // обратить последний массив, чтобы расположить
    // элементы jdoc в прямом порядке
    $template_tags_last = array_reverse($template_tags_last);

    $this->_template_tags = $template_tags_first + $template_tags_last;
  }
  return $this;
}
```

Ключевым для понимания кода данного метода является оператор `preg_match_all`. Этот оператор проще пояснить, чем составить. В нем создаются три массива. Первый массив содержит список всех элементов разметки `jdoc:include` из файла шаблона и размещается в элементе массива `$matches[0]`. Второй массив содержит список атрибутов `type`, обозначающих тип каждого из этих элементов, и размещается в элементе массива `$matches[1]`. К типам подобных элементов обычно относятся `head`, `modules`, `messages`, `debug` или `component`. И третий массив содержит список остальных атрибутов этих элементов, кроме атрибута `type`, и размещается в элементе массива `$matches[2]`.

На основании этой информации в цикле for сначала создается ассоциативный массив из элементов type, name и attribs и затем он сохраняется в поле _template_tags объекта $document, как показано в выражении $this->_template_tags.

А теперь можно приступить непосредственно к компоновке страницы, расположив отдельные ее части в нужных местах шаблона. Вернувшись к методу render() из класса JSite, инициируем сначала системное событие onBeforeRender, а затем проверим, кеширована ли страница. И далее выполним следующую строку кода:

```
JResponse::setBody($document->render($caching, $params));
```

В этой строке кода устанавливается поле объекта типа JResponse с помощью метода render() из класса JDocumentHTML. Исходный код этого метода приведен в листинге 3.17.

Листинг 3.17. Метод render() из класса JDocumentHTML

```
public function render($caching = false, $params = array())
{
  $this->_caching = $caching;

    if (!empty($this->_template))
    {
      $data = $this->_renderTemplate();
    }
    else
    {
      $this->parse($params);
      $data = $this->_renderTemplate();
    }

  parent::render();
  return $data;
}
```

Ключевая строка кода данного метода выделена в листинге 3.17 полужирным. В ней вызывается метод _renderTemplate(), циклически обращающийся к элементам массива, созданного ранее и сохраненного в поле $document->_template_tags объекта текущего документа, извлекая данные из этого массива, как выделено полужирным в листинге 3.18.

Листинг 3.18. Метод _renderTemplate() из класса JDocumentHTML

```
private function _renderTemplate() {
  $replace = array();
  $with = array();

  foreach($this->_template_tags AS $jdoc => $args)
  {
      $replace[] = $jdoc;
      $with[] = $this->getBuffer($args['type'], $args['name'],
 $args['attribs']);
  }

  return str_replace($replace, $with, $this->_template);
}
```

Ключевой в данном метода является следующая строка кода:

```
  $with[] = $this->getBuffer($args['type'], $args['name'],
$args['attribs']);
```

Напомним, что в поле _template_tags объекта текущего документа $document содержится массив элементов с информацией о компонентах, модулях, заголовках, сообщениях и отладке. Каждый из этих элементов обрабатывается в цикле foreach и загружается в массив $with.

Компонент уже воспроизведен, и поэтому он просто копируется из поля $contents. Элементы разметки jdoc:include для модулей обычно обрабатываются по их месту расположения в шаблоне. В таком случае каждый модуль выполняется по месту его назначения для данного пункта меню, а результат размещается в массиве $with. Аналогичным образом обрабатываются элементы head, messages и debug.

И в последней строке кода данного метода оператор замены символьных строк вставляет конкретный текст, выводимый для каждого модуля, компонента, сообщения, заголовка и элемента отладки в файл шаблона. На данном этапе элементы страницы форматируются в шаблоне и готовятся для отправки браузеру.

Важное следствие такого способа обработки состоит, в частности, в том, что модуль выполняется только в том случае, если он назначен на определенное место в шаблоне. Благодаря этому исключаются потери производительности для тех модулей, которые назначены на другие места или для других пунктов меню.

А теперь вернемся к упомянутому выше методу render() из класса JDocumentHTML и вызовем одноименный метод его родительского класса JDocument из файла libraries/joomla/document/document.php. Этот метод просто устанавливает видоизмененную дату и тип содержимого для заголовка документа, а затем возвращает переменную $data, которая содержит файл шаблона index.php с содержимым страницы, заменяющим элементы разметки jdoc:include.

Напомним, что рассмотрение данной части программы мы начинали со следующей строки кода из метода render() класса JSite:

```
JResponse::setBody($document->render($caching, $params));
```

На данном этапе мы завершили выполнение части document->render($caching, $params) оператора из приведенной выше строки кода. Осталось лишь передать результат выполнения этой части оператора методу JResponse::setBody(). В исходном коде данного метода, приведенном в листинге 3.19, получающийся в итоге текст просто размещается в поле $body данного класса.

Листинг 3.19. Метод setBody() из класса JResponse

```
public static function setBody($content)
{
  self::$body = array((string) $content);
}
```

И последнее, что требуется сделать в методе render(), — инициировать системное событие onAfterRender. В этот момент мы возвращаемся к файлу index.php из папки верхнего уровня.

Вывод страницы

Теперь нам осталось лишь отправить страницу браузеру. Соответствующий код приведен в самом конце файла index.php, как показано в листинге 3.20.

Листинг 3.20. Исходный код в конце файла index.php

```
// пометить состояние после воспроизведения в профилировщике
JDEBUG ? $_PROFILER->mark('afterRender') : null;

// возвратить ответ
echo $app;
```

Как и прежде, в операторе JDEBUG лишь помечается момент наступления события afterRender для профилирования приложения. Другое дело — последняя строка кода echo $app;. Напомним, что в переменой $app хранится объект типа JSite. А команда echo языка PHP служит для отправки текстового сообщения браузеру. Что же тогда означает передача объекта по команде echo?

Ответить на этот вопрос нетрудно. Если у объекта имеется специальный метод __toString(), он вызывается автоматически при выполнении команды echo с объектом. В противном случае команда echo лишь возвращает имя класса данного объекта.

Проанализировав состав класса JSite из файла includes/application.php, мы не обнаружим в нем метод __toString(). Но если сделать то же самое в отношении его родительского класса JApplication из файла libraries/joomla/application/application.php, то окажется, что в нем имеется метод __toString(), наследуемый классом JSite. Исходный код этого метода приведен в листинге 3.21.

Листинг 3.21. Метод __toString() из класса JApplication

```
public function __toString()
{
  $compress = $this->getCfg('gzip', false);

  return JResponse::toString($compress);
}
```

Этот метод, в свою очередь, вызывает метод toString() из класса JResponse. Исходный код метода toString() приведен в листинге 3.22.

Листинг 3.22. Метод toString() из класса JResponse

```
public static function toString($compress = false)
{
  $data = self::getBody();

  // не сжимать страницу, если сервер все равно собирается это сделать,
  // иначе это напрасная трата времени
  if ($compress && !ini_get('zlib.output_compression')
↳&& ini_get('output_handler')!='ob_gzhandler') {
    $data = self::compress($data);
  }

  if (self::allowCache() === false) {
```

```
        self::setHeader('Cache- Control', 'no-cache', false);
          // Протокол HTTP 1.0
        self::setHeader('Pragma', 'no-cache');
      }

      self::sendHeaders();

      return $data;
    }
```

Данный метод выполняет следующие действия.

1. Размещает видимую часть (так называемое тело) страницы в переменной `$data`.
2. Проверяет, следует ли сжать страницу или нет.
3. Проверяет наличие кеширования и соответственно устанавливает заголовок страницы.
4. Отправляет заголовки страницы.
5. Возвращает переменную `$data`, содержащую всю страницу.

Содержимое переменной `$data` затем возвращается команде `echo` языка PHP, которая отправляет его браузеру.

Краткие итоги сеанса работы с веб-сайтом Joomla

Итак, мы немало потрудились, чтобы отобразить страницу. Чему же мы в итоге научились?

На первых стадиях рассматриваемого здесь процесса мы лишь выяснили местоположение файлов, загрузили интегрированную среду Joomla и определили, следует ли начинать или продолжать сеанс работы с веб-сайтом. Затем мы обнаружили и выполнили компонент. После этого выполнили все модули и разместили все содержимое страницы в файле шаблона. И наконец, мы отправили ее браузеру. По ходу дела мы инициировали ряд системных событий, которые могут быть обработаны в подключаемых модулях.

И хотя на каждой стадии данного процесса нам пришлось воспользоваться немалым объемом кода, он, тем не менее, организован в относительно небольшое количество методов высокого уровня, каждый из которых способен, если потребуется, углубиться в методы низкого уровня. Методы высокого уровня структурированы таким образом, чтобы на каждой стадии данного процесса мы имели дело только с одним заданием. Блок-схема всего процесса приведена на рис. 3.9.

В этом кратком обзоре цикла выполнения команд мы могли также видеть, насколько просто архитектура Joomla позволяет внедрять расширения в интегрированную среду, предоставляемую системой CMS. Компоненты, модули, шаблоны и подключаемые модули выполняются во вполне определенные моменты данного процесса. При этом не делается никакого различия между основными расширениями, включенными в стандартную установку Joomla, и расширениями сторонних разработчиков. Все расширения интерпретируются одинаково, благодаря чему обеспечивается плавность их работы.

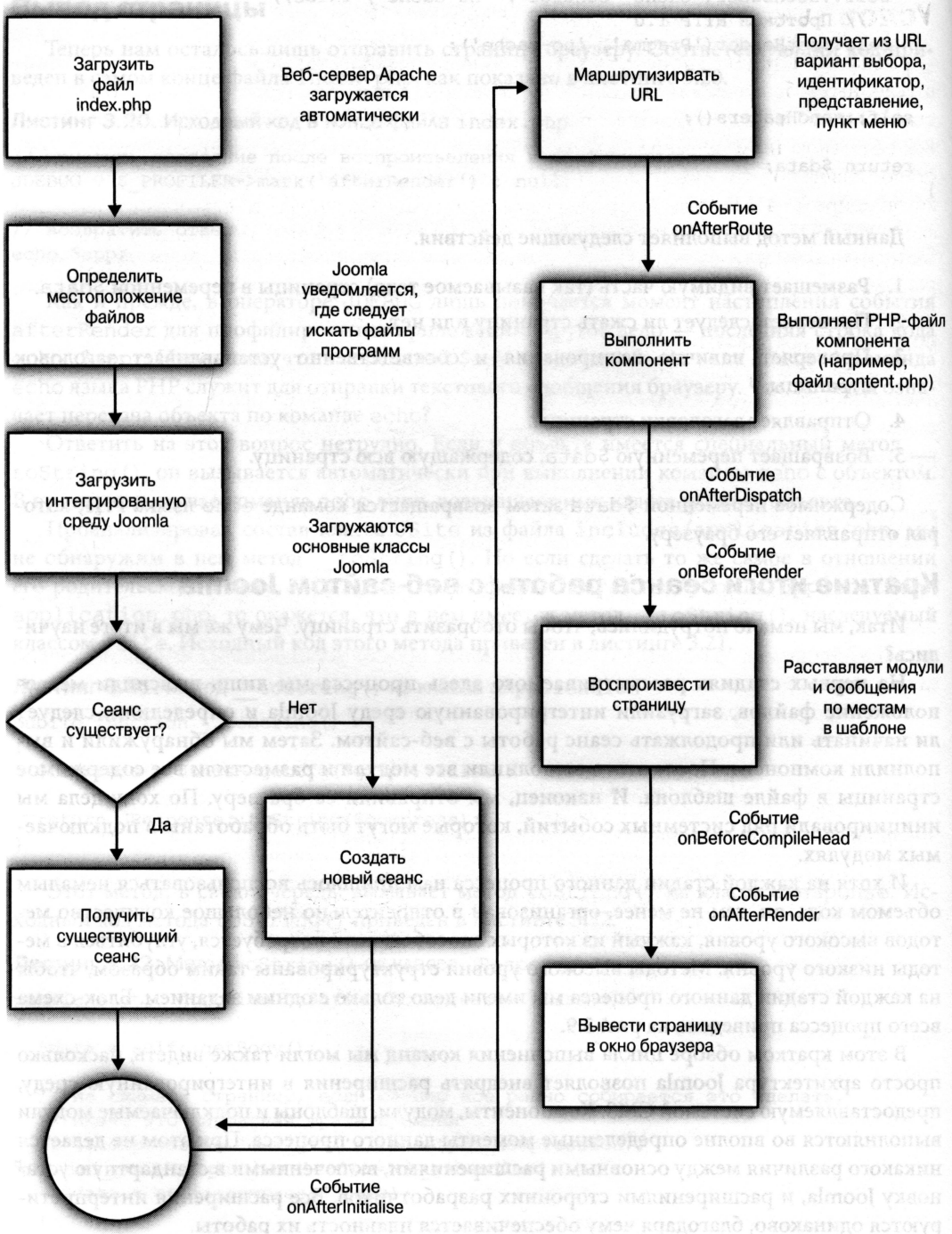


Рис. 3.9. Блок-схема цикла выполнения команд в Joomla

Условные обозначения имен

Одна из методик, упрощающих чтение и понимание кодовой базы Joomla, состоит в условных обозначениях имен файлов и папок. Характерный тому пример был представлен в предыдущем разделе. Напомним, что мы получили имя компонента из метода `route()` (в данном примере — `com_content`). И как только мы узнали это, то и Joomla стало "известно", что нужно искать файл с именем `content.php`. Подобная логика жестко закодирована в Joomla. Таким образом, для любого компонента нам заранее известен файл, который должен быть загружен, чтобы начать выполнение этого компонента.

Если известны эти условные обозначения, то намного проще исследовать существующие компоненты и понять, как создавать новые. Например, мы знаем, что нужно искать папку `com_имя_моего_компонента`, а в ней — файл `com_имя_моего_компонента.php`. Мы также знаем, что если в Joomla встретится URL, направляющий к варианту выбора `com_имя_моего_компонента`, то в итоге будет выполнен файл `com_имя_моего_компонента.php`. Такие условные обозначения приняты в Joomla повсеместно. И на них мы будем обращать внимание при рассмотрении различных типов расширений в последующих главах.

Глобальные объекты

Некоторые объекты используются в стольких местах кода Joomla, что было бы целесообразно, чтобы они были легко доступны постоянно. Как упоминалось ранее, класс `JFactory` из файла `libraries/joomla/factory.php` загружается в тот момент, когда сценарий из файла `libraries/joomla/import.php` загружается по сценарию из файла `includes/framework.php`. Класс `JFactory` обеспечивает быстрый доступ к целому ряду объектов практически из любого места в программе. Все эти объекты перечислены в табл. 3.11.

Таблица 3.11. Глобальные объекты

Тип объекта	Описание	Код доступа
`Japplication` `JSite` `Jadministrator`	Объект приложения; `Jsite` в пользовательской части; `Jadministrator` в административной части	`$app = JFactory::getApplication();`
`Jregistry`	Значения из файла `configuration.php` в форме объекта типа `JRegistry`	`$config = JFactory::getConfig();`
`Jsession`	Текущий сеанс работы с веб-сайтом	`$session = JFactory::getSession();`
`Jlanguage`	Объект языка	`$language = JFactory::getLanguage();`
`Jdocument`, `JdocumentHTML`, `JdocumentFeed`, `JdocumentJSON`	Текущий документ	`$document = JFactory::getDocument();`
`Juser`	Текущий пользователь	`$user = JFactory::getUser();`

Окончание табл. 3.11

Тип объекта	Описание	Код доступа
Jaccess	Текущий объект доступа (используется при контроле доступа)	`$access = JFactory::getACL();`
Jdatabase	Объект базы данных, используемый для выполнения запросов	`$dbo = JFactory::getDbo();`
Jmail	Почтовый объект для отправки электронной почты с веб-сайта	`$mail = JFactory::getMailer();`
JXMLElement	Элемент разметки XML	`$xml = JFactory::getXML($pathToXml)`
Jeditor	Объект редактора	`$editor = JFactory::getEditor();`
JURI	Объект URI	`$uri = JFactory::getUri();`
JDate	Объект даты и времени	`$date = JFactory::getDate();`

Кроме класса JDate, каждый из классов, перечисленных в табл. 3.11, должен иметь лишь один объект данного класса. Например, в пользовательской части веб-сайта должен быть только один объект приложения типа JSite, а в административной части — только один аналогичный объект типа JAdministrator. Аналогично должен быть только один объект типа JSession, JUser, JDatabase и т.д.

Для подобных случаев в объектно-ориентированном программировании нередко служат шаблоны проектирования, называемые "одиночкой" и "фабрикой" и обеспечивающие существование лишь одного объекта данного типа в любой заданный момент времени. И действуют они по очень простому принципу.

Рассмотрим этот принцип на примере применения объекта приложения. В классе JFactory содержится поле $application, где хранится один объект приложения, которым нужно пользоваться в течение всего цикла выполнения команд. И когда потребуется объект приложения, он может быть получен по команде JFactory::getApplication().

При выполнении этой команды в классе JFactory сначала проверяется, имеется ли уже объект приложения в поле $application. Если этот объект имеется, то он просто возвращается из данного поля. А если поле $application пустое, это означает, что в данном цикле мы впервые пытаемся получить доступ к объекту приложения. В таком случае мы создаем новый объект приложения, сохраняем его в поле $application из класса JFactory, а затем возвращаем его из данного поля. Таким образом, единственный объект приложения подготавливается в поле $application из класса JFactory для последующего применения в любой момент, когда он может потребоваться.

Краткий обзор таблиц базы данных

База данных (как правило, MySQL) применяется в Joomla для хранения всех данных на веб-сайте. Стандартная база данных для версии Joomla 2.5 состоит из 34 таблиц. В этом разделе приводится краткое описание каждой таблицы базы данных и ее назначение.

О префиксах имен таблиц

При установке Joomla имеется возможность выбрать префикс для имен таблиц в базе данных. Для доступа к таблице в коде PHP обычно служит обобщенный префикс `#__`. Метод `query()` из класса `JDatabaseMySQLi` вызывает метод `replacePrefix()`, заменяющий обобщенный префикс на конкретный префикс, выбираемый из файла `configuration.php`.

Ради простоты в именах таблиц, упоминаемых в этой книге, употребляется префикс `#__`, например `#__content` (табл. 3.12). Но в именах таблиц реальной базы данных употребляется префикс, введенный при установке Joomla. Некоторые таблицы связываются с классом `JTable`. Это обычно делается для целей редактирования столбцов таблицы пользователем. Достаточно создать класс `JTable` для таблицы, и большая часть кода редактирования столбцов этой таблицы будет подготовлена автоматически. Подробнее об этом речь пойдет при создании специального компонента в главе 9.

Таблица 3.12. Перечень таблиц базы данных в Joomla

Таблица/Класс языка PHP	Описание
`#__assets` / `JTableAsset`	Это новая таблица, введенная для системы списков контроля доступа (ACL) в версии 1.6. Она содержит строки для каждого компонента, также для каждого элемента с полномочиями ACL. К их числу относятся строка `Root Asset` (Корневой ресурс) для глобальных прав доступа, строка для каждой категории компонентов, а также строка для каждой статьи. В столбце `rules` (правила) хранятся групповые права доступа в формате JSON. Эта таблица используется всякий раз, когда требуется проверить, уполномочен ли пользователь выполнять конкретное действие
`#__associations` / Отсутствует	Используется для собственного многоязычного средства в Joomla, чтобы дать возможность связать отдельный пункт меню с соответствующим пунктом меню на другом языке мира
`#__banners` / Отсутствует	Содержит строки для каждого баннера, определенного на веб-сайте
`#__banner_clients` / Отсутствует	Содержит строки для каждого клиента баннера, определенного на веб-сайте
`#__banner_tracks` / Отсутствует	Содержит строки для каждого канала отслеживания баннеров, определенного на веб-сайте
`#__categories` / `JTableCategory`	Содержит строки для каждой категории, определенной на веб-сайте, включая статью, баннер, ленту новостей и веб-ссылку
`#__contact_details` / Отсутствует	Содержит строки для каждой контактной информации, определенной на веб-сайте
`#__content` / `JTableContent`	Содержит строки для каждой статьи, определенной на веб-сайте
`#__content_frontpage` / Отсутствует	Содержит строки для каждой статьи, назначенной для пункта меню блога по избранным статьям

Продолжение табл. 3.12

Таблица/Класс языка PHP	Описание
#__content_rating / Отсутствует	Содержит строки для каждой статьи, оцененной на веб-сайте
#__core_log_searches / Отсутствует	Содержит строки для каждого критерия поиска, зарегистрированного на веб-сайте
#__extensions / JTableExtension	Содержит строки для каждого расширения, установленного на веб-сайте, включая компоненты, библиотеки, модули, шаблоны, подключаемые модули, языки и файлы
#__finder_filters / FinderTableFilter	Содержит строки для каждого фильтра поиска, установленного по команде Smart Search⇨Manage Search Filters (Интеллектуальный поиск⇨Управлять фильтрами поиска)
#__finder_links / FinderTableLink	Содержит строки для каждого элемента содержимого, индексированного средствами интеллектуального поиска
#__finder_links_terms(0-f) / Отсутствует	Сопоставляет критерии поиска со ссылками, учитывая весовой коэффициент. (Следует иметь в виду, что 15 таблиц, по существу, составляют единую логическую таблицу, но разделены на отдельные таблицы ради ускорения их индексирования.)
#__finder_taxonomy / FinderTableMap	Содержит строки для элементов каждого типа, включая категорию, язык, регион, страну и т.д.
#__finder_taxonomy_map / Отсутствует	Сопоставляет ссылки с элементами классификации
#__finder_terms / Отсутствует	Содержит строки для каждого поискового элемента, индексированного по всем элементам содержимого
#__finder_terms_common / Отсутствует	Содержит строки для каждого общеупотребительного в языке слова (не подлежит индексированию)
#__finder_tokens, #__finder_tokens_aggregate / Отсутствует	Временные таблицы, используемые в процессе индексирования
#__finder_types / Отсутствует	Содержит строки для содержимого каждого типа, включая категорию, статью, ленту новостей, веб-ссылку и т.д.
#__languages / JTableLanguage	Содержит строки для каждого языка мира, установленного на веб-сайте
#__menu / JTableMenu	Содержит строки для каждого пункта меню, определенного в пользовательской и административной частях веб-сайта
#__menu_types / JTableMenuType	Содержит строки для каждого меню, определенного в пользовательской части веб-сайта
#__messages / Отсутствует	Содержит строки для каждого частного сообщения, отправляемого на веб-сайте
#__messages_cfg / Отсутствует	Содержит строки для каждого пользователя административной части, настраивающего веб-сайт по команде Components⇨Messaging⇨My Settings (Компоненты⇨Обмен сообщениями⇨Мои настройки)

Окончание табл. 3.12

Таблица/Класс языка PHP	Описание
#__modules / JTableModule	Содержит строки для каждого модуля, определенного в системе. Значение **0** в столбце client_id обозначает модуль, относящийся к пользовательской части, а значение **1** — модуль, относящийся к административной части веб-сайта
#__modules_menu / Отсутствует	Это таблица соответствий, в которой показано назначение модулей для отдельных пунктов меню. Назначение модулей для отдельных пунктов меню указывается в столбце menuid следующим образом: значение **0** обозначает, что модуль назначен для всех пунктов меню; положительное значение — для данного конкретного пункта меню; отрицательное значение — для всех пунктов меню, кроме данного пункта
#__newsfeeds / Отсутствует	Содержит строки для каждой ленты новостей, созданной на веб-сайте
#__redirect_links / Отсутствует	Содержит строки для каждой переадресации, организованной на веб-сайте
#__schemas / Отсутствует	Содержит строки для каждого расширения, внесшего изменения в базу данных во время установки, наряду с последней установленной версией расширения
#__session / JTableSession	Содержит строки для каждого активного сеанса работы с веб-сайтом
#__template_styles / Отсутствует	Содержит строки для каждого стиля шаблона, определенного на веб-сайте
#__updates / JTableUpdate	Содержит строки для каждого доступного для установки пакета
#__update_categories / Отсутствует	Эта таблица служит для разделения обновлений на отдельные категории. Она сопровождается в Joomla автоматически
#__update_sites / Отсутствует	Содержит список сайтов обновления, выбираемый из XML-файла для каждого расширения
#__update_sites_extensions / Отсутствует	Это таблица соответствий, связывающая расширения #__ с обновлениями #__ и содержащая строки для каждого сочетания расширения и сайта обновления, где это расширение может быть обновлено
#__usergroups / JTableUsergroup	Содержит строки для каждой группы пользователей, определенной на веб-сайте
#__users / JTableUser	Содержит строки для каждого пользователя, определенного на веб-сайте
#__user_profiles / Отсутствует	Содержит строки для каждой группы, членом которой является данный пользователь
#__viewlevels / JTableViewlevel	Содержит строки для каждого уровня представления, определенного на веб-сайте
#__weblinks / Отсутствует	Содержит строки для каждой веб-ссылки, определенной на веб-сайте

Резюме

В этой главе вы ознакомились с внутренним механизмом работы Joomla. Сначала в ней была исследована структура папок и файлов в пользовательской и административной частях веб-сайта Joomla. Затем был рассмотрен ряд общих понятий, которыми веб-программирование отличается от традиционного программирования настольных приложений, включая необходимость организации защиты от нарушителей типа хакеров.

После этого был подробно проанализирован весь код, управляющий типичным циклом выполнения команд в Joomla. Попутно было показано, каким образом принятые в Joomla условные обозначения имен упрощают задачу разработчиков.

И в конце было показано, как находить глобальные объекты при написании кода, а также приведено краткое описание таблиц базы данных, входящей в стандартную установку Joomla.

4

Расширение Joomla замещениями компоновки

Желаете ли вы изменить способ отображения какого-нибудь содержимого на вашем веб-сайте Joomla, добавить или удалить некоторую информацию, сделать нечто необычное, чтобы страница выглядела именно так, как вам хочется? Если желаете, то эта глава написана именно для вас!

Joomla позволяет чрезвычайно просто изменить способ отображения содержимого каждой страницы веб-сайта. Более того, это можно сделать, не особенно вникая в особенности компоновки информации на странице и едва ли вообще умея программировать на языке PHP.

Как будет показано в этой главе, достаточно скопировать или произвести элементарную правку одного или нескольких файлов, чтобы определить порядок отображения информации на странице. Эта глава начинается с самых основ применения шаблонов в Joomla. Затем в ней поясняется принцип действия замещений компоновок, т.е. кода, как по мановению волшебной палочки изменяющего внешний вид страниц веб-сайта. Далее будут рассмотрены примеры, демонстрирующие различные виды замещений компоновок, начиная с элементарного и кончая более сложными примерами. И в конце главы будут представлены другие элементы оформления веб-страниц, помимо компоновок, которые можно замещать в Joomla.

Основы применения шаблонов

Содержимое представляет собой информацию, отображаемую на веб-странице. В представление этой информации входит все, что касается ее отображения, в том числе компоновка страницы, выбор шрифта, цвета и любой графики (прямоугольных рамок, линий, изображений и логотипов). Главная цель Joomla — дать возможность вносить любые изменения в содержимое и представление страниц веб-сайта независимым образом. И достигается эта цель с помощью шаблонов.

Шаблон является разновидностью расширения, определяющего все особенности представления веб-сайта. В главе 2 было вкратце продемонстрировано, что для полного изменения внешнего вида веб-сайта достаточно внести коррективы в используемый по умолчанию шаблон. А в данном разделе будет подробно пояснено, как это делается.

Следует, однако, иметь в виду, что мы не будем здесь вдаваться в подробности написания шаблонов для Joomla. Это довольно обширная тема, выходящая за рамки данной книги. Но материал этого раздела поможет лучше понять назначение и принцип действия шаблонов с точки зрения разработки веб-сайтов.

Папки и файлы шаблонов

По умолчанию для пользовательской части веб-сайта Joomla выбирается шаблон `beez_20`. Если заглянуть в папку `templates/beez_20`, то в ней можно обнаружить папки и файлы, показанные на рис. 4.1.

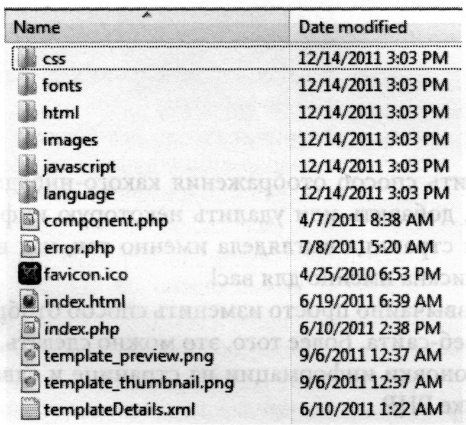


Рис. 4.1. Папки и файлы шаблона `beez_20`

В папке `css`, как и следовало ожидать, находятся вложенные таблицы стилей CSS, подобранные для данного шаблона. Аналогично в папке `fonts` содержатся любые специальные шрифты, необходимые для шаблона.

Папка `html` является именно тем местом, где хранятся файлы любых замещений компоновок. Для шаблона `beez_20` в этой папке имеется лишь один файл замещения под названием `modules.php`. Это файл специальной стилевой рамки модуля. Под *стилевой рамкой модуля* подразумеваются функции языка PHP, обеспечивающие дополнительную HTML-разметку, в которую помещается сам модуль. Такая рамка иногда еще называется *оболочкой* для вывода модуля. Более подробно стилевая рамка модуля рассматривается далее в этой главе.

Если заглянуть в папку `templates/beez5/html`, то в ней можно обнаружить папки `com_contact` и `com_content`. Эти папки содержат файлы замещений компонентов Contacts (Контакты) и Content (Содержимое). Принцип действия замещений будет рассмотрен далее в этой главе, а до тех пор достаточно знать, что файл замещения компоновки должен быть помещен в нужную папку в шаблоне, чтобы он был найден и использован в Joomla.

В папке images содержатся изображения, отобранные для шаблона, в том числе логотипы, кнопки, пиктограммы и т.д. А в папке javascript находятся любые файлы сценариев на языке JavaScript, требующиеся для шаблона. Так, для шаблона beez_20 имеются два файла сценариев JavaScript: один обеспечивает общее функционирование этого шаблона, а другой — дает пользователю возможность изменить размер шрифта.

Настройка параметров шаблона выполняется в административной части веб-сайта Joomla с помощью компонента Template Manager (Диспетчер шаблонов). Для этого требуются файлы языковой поддержки, с помощью которых выводимый на экран текст переводится на выбранный язык мира. А сами файлы языковой поддержки хранятся в папке language.

Ниже всех этих папок в папке шаблона располагаются файлы, перечисленные в табл. 4.1.

Таблица 4.1. Описание файлов шаблона

Файл	Описание
component.php	Шаблон для печати или отправки электронной почтой содержимого компонента (например, статьи)
error.php	Шаблон для отображения ошибочных страниц (например, при появлении недействительного URL)
favicon.ico	Пиктограмма, отображающаяся в окне браузера рядом с URL
index.html	Пустой файл, не позволяющий пользователю просматривать папку
index.php	Главный файл шаблона, управляющий расположением модулей, компонентов и сообщений
template_preview.png	Предварительно просматриваемая графика с изображением шаблона
template_thumbnail.png	Миниатюрная графика с изображением шаблона
templateDetails.xml	XML-файл со списком всех папок, файлов и мест расположения в шаблоне; определяет также любые параметры настройки шаблона

Файл шаблона index.php

В Joomla принято, чтобы шаблон содержал файл index.php, который, по существу, служит шаблоном для страниц веб-сайта. (Главный файл шаблона может называться как угодно в зависимости от значения переменной tmpl в запросе. По умолчанию в качестве этого значения выбирается index.php, если переменная tmpl не определена.) Файл index.php содержит схематическое представление страницы с пустыми местами для заполнения конкретным содержимым каждой страницы.

В главе 3 пояснялось, каким образом страница веб-сайта Joomla компонуется в методе render(). В частности, программа сначала составляет список всех элементов jdoc:include в файле index.php шаблона, а затем заполняет пустые места соответствующим содержимым каждого элемента.

В качестве примера рассмотрим файл index.php для шаблона atomic. Этот шаблон входит в стандартный вариант распространения Joomla и представляет собой простейший шаблон, служащий в качестве отправной точки для создания новых шаблонов.

Если вы еще не ознакомились с основами программирования на PHP, то теперь самое время обратиться к материалу приложения А, где приводится краткий курс ООП и PHP.

Файл `index.php` служит отличным примером того, насколько просто код PHP сочетается с кодом HTML средствами языка PHP. В первой части этого файла, приведенной в листинге 4.1, находится единственный фрагмент кода в одном элементе разметки PHP.

Листинг 4.1. Открывающий элемент разметки PHP в начале файла `index.php` шаблона `atomic`

```
<?php
/**
 * @package Joomla.Site
 * @copyright     Copyright (C) 2005 - 2012 Open Source Matters, Inc.
↳All rights reserved.
 * @license       GNU General Public License version 2 or later; see
↳LICENSE.txt
 */

defined('_JEXEC') or die;

/* В следующей строке кода загружается библиотека MooTools языка JavaScript */
JHTML::_('behavior.framework', true);

/* В следующей строке кода получается объект приложения для таких функций,
  как отображение наименования сайта */
$app = JFactory::getApplication();
?>
```

В этом фрагменте кода гарантируется, что все остальное будет выполняться в среде Joomla, загружается библиотека интегрированной среды MooTools языка JavaScript, а затем получается экземпляр объекта типа `JApplicationSite`. Подробнее об интегрированной среде MooTools в частности и языке JavaScript вообще речь пойдет в главе 12.

В следующем фрагменте кода, приведенном в листинге 4.2, демонстрируется ряд очень важных особенностей шаблонов.

Листинг 4.2. Второй фрагмент кода из файла `index.php` шаблона `atomic`

```
<?php echo '<?'; ?>xml version="1.0" encoding="<?php echo $this->_charset
↳?>" ?>
<!DOCTYPE html PUBLIC "- //W3C//DTD XHTML 1.0 Strict//EN"
↳"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
↳xml:lang="<?php echo $this->language;?>"
↳lang="<?php echo $this->language; ?>" dir="<?php echo $this->direction;
↳?>" >
  <head>
    <!-- В следующем дескрипторе JDOC Head загружается весь заголовок
         и его метаданные из настроек и содержимого вашего сайта. -->
    <jdoc:include type="head" />

    <!-- В следующих пяти строках кода загружается среда Blueprint CSS
         Framework (http://blueprintcss.org). Если вы не собираетесь
         пользоваться данной средой, удалите эти строки кода. -->
```

```
    <link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/
↳<?php echo $this->template ?>/css/blueprint/screen.css" type="text/css"
↳media="screen, projection" />
    <link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/
↳<?php echo $this->template ?>/css/blueprint/print.css" type="text/css"
↳media="print" />
    <!-- [if lt IE 8]><link rel="stylesheet" href="blueprint/ie.css"
         type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/
↳<?php echo $this->template ?>/css/blueprint/plugins/joomlaKnav/
screen.
css"
↳type="text/css" media="screen" />
↳<?php echo $this->template ?>/css/blueprint/plugins/joomla-nav/
↳screen.css" type="text/css" media="screen" />

    <!-- В следующей строке кода загружается файл вложенных таблиц
         стилей CSS шаблона, находящийся в папке шаблона. -->
    <link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/
↳<?php echo $this->template ?>/css/template.css" type="text/css" />

    <!-- В следующих четырех строках кода загружается среда Blueprint CSS
         Framework и файл вложенных таблиц стилей CSS шаблона для языков с
         левосторонним письмом. Если вы не собираетесь пользоваться данными
         средствами, удалите эти строки кода. -->
    <?php if($this->direction == 'rtl') : ?>
      <link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/
↳<?php echo $this->template ?>/css/blueprint/plugins/rtl/screen.css"
↳type="text/css" />
      <link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/
↳<?php echo $this->template ?>/css/template_rtl.css" type="text/css" />
    <?php endif; ?>

    <!-- В следующей строке кода из папки шаблона загружается файл
         сценария JavaScript для шаблона. По умолчанию он пуст. -->
    <script type="text/javascript" src="<?php echo $this->baseurl ?>/
↳templates/<?php echo $this->template ?>/js/template.js"></script>
  </head>
```

Во-первых, следует заметить, что в рассматриваемый здесь файл входит объявление `DOCTYPE` типа HTML-документа. Это означает, что тип документа полностью определяется шаблоном. И в этом есть определенный смысл, если учесть, что шаблон управляет всем представлением веб-сайта. Для того чтобы убедиться в этом на практике, перейдите к диспетчеру шаблонов в Joomla, откройте исходный стиль шаблона `Beez5 - Default` и измените значение параметра `html version` с `xhtml` на `html5`. Затем отобразите страницу `Fruit Shop` (Фруктовая лавка) и просмотрите исходный код данной страницы. В нем вы обнаружите дескриптор `<!DOCTYPE html>` вместо дескриптора `<!DOCTYPE html PUBLIC "- //W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`.

Во-вторых, обратите внимание на то, что файл шаблона содержит элементы `meta`, `title` и `script` в дескрипторе `head` типа HTML-заголовка. Эти элементы можно изменять

динамически в коде, если есть такая потребность. Например, элемент `title` часто изменяется для каждой страницы веб-сайта, но другие элементы могут быть определены здесь и использованы без всяких изменений на всех страницах веб-сайта.

И в-третьих, следует иметь в виду, что данный файл является обычным файлом исходного кода PHP, а следовательно, в нем можно выполнять любые команды языка PHP. И это дает немало возможностей для интересного экспериментирования с шаблоном, как будет продемонстрировано на конкретном примере далее в этой главе.

Сочетание элементов разметки PHP и HTML

Язык PHP нашел широкое применение в веб-разработке, в частности, потому, что он позволяет без труда сочетать элементы PHP и HTML в любом файле с расширением `.php`. В качестве примера рассмотрим следующую строку кода из файла `index.php`:

```
<link rel="stylesheet" href="<?php echo $this->baseurl ?>/templates/
↳<?php echo $this->template ?>/css/blueprint/screen.css"
↳type="text/css" media="screen, projection" />
```

Это всего лишь элемент HTML-ссылки, но с динамическими возможностями PHP. Атрибут `href` содержит два элемента разметки PHP, вычисляемые интерпретатором PHP и преобразуемые в соответствующие им текстовые значения, которые затем вставляются в разметку HTML-документа так, как будто они набраны вручную.

В данном примере элемент разметки `<?php echo $this->baseurl ?>` преобразуется в значение переменной `$this->baseurl` в коде PHP, а элемент разметки `<?php echo $this->template ?>` — в значение переменной `$this->template`.

Так, если переменная `$this->baseurl` содержит значение `"http://myjoomlasite.org"`, а переменная `$this->template ?>` —значение `"atomic"`, то в результате выполнения приведенной выше строки кода будет получен следующий (выделенный полужирным) текст ссылки, как если бы он был набран вручную:

```
<link rel="stylesheet"
href="http://myjoomlasite.org/templates/atomic/css/blueprint/screen.css"
type="text/css" media="screen, projection" />
```

Во многих местах приведенного выше кода из файла `index.php` можно обнаружить элементы разметки PHP, вставленные в элементы разметки HTML. Дополнительные сведения о коде и синтаксисе языка PHP приведены в приложении А.

Места расположения в шаблонах

Рассмотрим оставшуюся часть файла `index.php` из шаблона `atomic`. Соответствующий код приведен в листинге 4.3.

Листинг 4.3. Последний кодовый блок из файла `index.php` шаблона `atomic`

```
<body>
  <div class="container">
    <hr class="space" />
    <div class="joomla-header span- 16 append-1">
      <h1><?php echo $app->getCfg('sitename'); ?></h1>
```

```
    </div>
    <?php if($this->countModules('atomic-search')) : ?>
      <div class="joomla-search span-7 last">
        <jdoc:include type="modules" name="atomic- search" style="none"/>
      </div>
    <?php endif; ?>
  </div>
  <?php if($this->countModules('atomic-topmenu')) : ?>
    <jdoc:include type="modules" name="atomic-topmenu" style="container"/>
  <?php endif; ?>

  <div class="container">
    <div class="span-16 append-1">
    <?php if($this->countModules('atomic- topquote')) : ?>
      <jdoc:include type="modules" name="atomic-topquote" style="none"/>
    <?php endif; ?>
    <jdoc:include type="message" />
    <jdoc:include type="component" />
    <hr />
    <?php if($this->countModules('atomic-bottomleft')) : ?>
      <div class="span-7 colborder">
        <jdoc:include type="modules" name="atomic-bottomleft"
          style="bottommodule" />
      </div>
      <?php endif; ?>

      <?php if($this->countModules('atomic-bottommiddle')) : ?>
      <div class="span-7 last">
        <jdoc:include type="modules" name="atomic-bottommiddle"
          style="bottommodule" />
      </div>
    <?php endif; ?>
    </div>
    <?php if($this->countModules('atomic-sidebar')) : ?>
      <div class="span-7 last">
      <jdoc:include type="modules" name="atomic-sidebar" style="sidebar"/>
      </div>
    <?php endif; ?>

    <div class="joomla-footer span-16 append-1">
      <hr />
      &copy;<?php echo date('Y'); ?> <?php echo $app->getCfg('sitename'); ?>
    </div>
      </div>
</body>
</html>
```

Не считая элементов разметки PHP, приведенный выше фрагмент кода можно рассматривать как обычную разметку статического HTML-документа, в которой определен элемент разметки `body`, а внутри него — несколько элементов разметки `div`. Обратите внимание на то, что в четвертой строке кода выполняются те же самые действия, что и прежде: сначала получается значение переменной `$app->getCfg('sitename')`, а затем оно вставляется в элемент разметки `h1`.

В этой части, завершающей файл index.php шаблона atomic, необходимо уяснить следующее. Во-первых, в ней имеется целый ряд элементов jdoc:include разметки HTML-документа. Но элементы разметки jdoc:include не являются стандартным для HTML, а специфическими для Joomla. Как пояснялось в главе 3, эти элементы служат в качестве мест заполнения, заменяемых в Joomla текстом компонентов, сообщений или модулей. Место расположения элементов разметки jdoc:include определяет расположение текста каждой части страницы. В средине рассматриваемого здесь фрагмента находятся следующие строки кода:

```
<jdoc:include type="message" />
<jdoc:include type="component" />
```

В этих строках кода указывается место, где средствами Joomla должны быть расположены сообщения и компонент. В Joomla допускается размещать только один компонент на странице, и поэтому для подстановки компонентов в данном фрагменте кода имеется единственный элемент разметки jdoc:include. Аналогично сообщения отображаются лишь в одном месте, хотя на страницу их можно выводить сколько угодно. Сообщение может уведомлять о завершении некоторого действия, предупреждении или ошибке.

Если проанализировать элемент разметки jdoc:include для подстановки модуля, то в нем можно обнаружить два дополнительных атрибута, как показано ниже.

```
<jdoc:include type="modules" name="atomic-search" style="none" />
```

Атрибут name определяет имя места расположения модуля в шаблоне. При создании модуля в компоненте Module Manager (Диспетчер модулей) это имя присваивается соответствующему месту расположения данного модуля. Когда же страница воспроизводится средствами Joomla, все модули обнаруживаются по месту их расположения в шаблоне и затем выполняются. А результат выполнения модуля, как, впрочем, и компонента или вывода сообщений, подставляется вместо соответствующего элемента разметки jdoc:include.

Атрибут style определяет так называемую стилевую рамку модуля, буквально называемую *хромированием модуля* (от англ. *module chrome*). В прошлом автомобили специально отделывались хромом в качестве украшения, и поэтому данный термин выбран для обозначения декоративной рамки, в которую вставляется веб-страница. Следовательно, стилевая рамка указывает Joomla обрамление и художественное оформление модуля. Подробнее об этом речь пойдет ниже, в разделе "Стилевая рамка модуля".

Альтернативный синтаксис условных операторов if/then/else в языке PHP

Как пояснялось ранее, одной из самых примечательных особенностей языка PHP является возможность сочетать элементы разметки PHP с элементами разметки HTML в одном и том же файле. Для этой цели в языке PHP предусмотрен альтернативный способ написания условных операторов if/then/else.

В качестве примера рассмотрим следующий фрагмент кода, взятый из файла index.php шаблона atomic:

```
<?php if($this->countModules('atomic-bottomleft')) : ?>
  <div class="span-7 colborder">
    <jdoc:include type="modules" name="atomic-bottomleft"
```

```
↳style="bottommodule" />
  </div>
<?php endif; ?>
```

Логика выполнения этого фрагмента кода подразумевает вычисление выражения в круглых скобках после условного оператора `if`, как показано ниже.

```
$this->countModules('atomic- bottomleft')
```

Если результатом вычисления этого выражения оказывается логическое значение `true`, то строки кода, расположенные между элементами `if` и `endif` данного условного оператора, посылаются браузеру. В противном случае эти строки кода просто игнорируются.

Условный оператор `if` является автономным элементом языка PHP, что позволяет вставлять обычные элементы разметки HTML-документа между элементами синтаксиса `if` и `endif` этого условного оператора (в данном случае — это элемент разметки `div`, содержащий элемент разметки `jdoc:include`). Этот же фрагмент кода можно было бы написать на PHP следующим образом:

```
<?php if($this->countModules('atomic-bottomleft')) {
  echo '<div class="span-7 colborder">';
  echo '<jdoc:include type="modules" name="atomic-bottomleft"
↳style="bottommodule" />';
  echo '</div>';
} ?>
```

Оба приведенных выше фрагмента кода выводят один и тот же текст. Но первый из них обладает тем преимуществом, что всякий, знакомый с HTML, может легко прочитать строки кода HTML в этом фрагменте кода PHP. А кроме того, эти строки кода читаются и проверяются на достоверность в HTML-редакторе. Но самое интересное, что тот же самый код можно написать и третьим способом, как показано ниже.

```
<?php if($this->countModules('atomic-bottomleft')) { ?>
  <div class="span-7 colborder">
    <jdoc:include type="modules" name="atomic-bottomleft"
↳style="bottommodule" />
  </div>
<?php } ?>
```

Единственное отличие этого третьего фрагмента кода от первого состоит в использовании фигурных скобок в открывающем (вместо двоеточия) и закрывающем дескрипторах PHP. И этот синтаксис позволяет вставлять элементы разметки HTML в блок управляющего оператора PHP, но он менее удобочитаем, чем в первом примере. Поэтому рекомендуется синтаксис из первого примера, тем более что именно он применяется в Joomla.

Аналогичный синтаксис имеется и для управляющих операторов языка PHP, в том числе `switch`, а также для циклов `for` и `foreach`. И в этих операторах он дает те же самые преимущества: возможность вставлять обычные элементы разметки HTML в блок управляющего оператора PHP. Другие примеры применения альтернативного синтаксиса приведены в приложении А.

Как же выбрать подходящий синтаксис? Очень просто. Большинство файлов в Joomla содержат один элемент разметки PHP и ни одного элемента разметки HTML, поскольку они не предназначены для непосредственного вывода текста в окно браузера. В таких файлах всегда используется обычный синтаксис языка PHP с фигурными скобками. А файлы, содержащие элементы разметки HTML, относятся к шаблонам и компоновкам веб-страниц. И в таких файлах, как правило, используется альтернативный синтаксис языка PHP.

Следует, однако, иметь в виду, что для интерпретатора языка PHP совершенно не важно, каким синтаксисом мы пользуемся. Но выбираем мы его из соображений удобочитаемости кода. И это справедливо вообще для многих вариантов решений, выбираемых при проектировании или программировании. Ведь самое главное, чтобы код был удобочитаем и понятен всем заинтересованным лицам.

К числу общеизвестных истин в разработке программного обеспечения относится то обстоятельство, что однажды выпущенная программа почти всегда эксплуатируется дольше, чем предполагают ее разработчики. А это означает, что код программы кому-то придется еще долго читать и сопровождать после того, как ее разработчики перейдут к другим проектам. Именно поэтому мы всегда стараемся писать код так, чтобы он был легко понятен другим.

И это тем более важно для такого программного обеспечения с открытым кодом, как Joomla и большинство расширений Joomla. Ведь мы не можем заранее предугадать, где именно будет использоваться открытый код. Но мы можем и должны постараться сделать его как можно более ясным, логичным и понятным, чтобы другим было легче им пользоваться.

Параметры настройки шаблонов

После регистрации в административной части веб-сайта Joomla, выбора сначала команды Extensions⇨Template Manager (Расширения⇨Диспетчер шаблонов), а затем исходного шаблона `Beez2 - Default` для редактирования стиля его оформления в правой части экрана появляются дополнительные параметры настройки, приведенные на рис. 4.2.

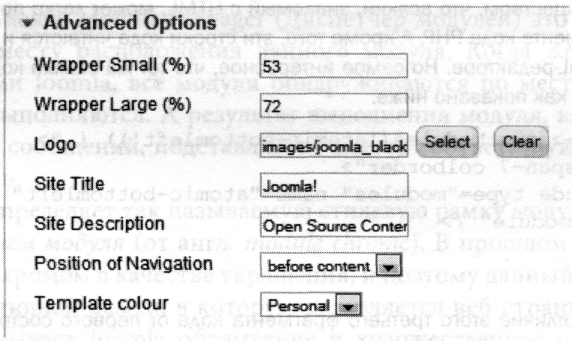


Рис. 4.2. Параметры настройки шаблона `beez_20`

Это лишь первый из многих примеров параметров, доступных для настройки в Joomla. Посмотрим, откуда они берутся, как используются и сохраняются в базе данных.

Параметры определяются в XML-файлах. В данном случае имеется в виду файл `templates/beez_20/templateDetails.xml`, в котором содержится и другая информация о шаблоне. Часть этого файла, где находится информация о параметрах, располагается в элементе `config`, как показано в листинге 4.4.

Листинг 4.4. Определения параметров шаблона `beez_20`

```
<config>
  <fields name="params">
    <fieldset name="advanced">
          <field name="wrapperSmall" class="validate- numeric"
```

```
type="text" default="53"
         label="TPL_BEEZ2_FIELD_WRAPPERSMALL_LABEL"
         description="TPL_BEEZ2_FIELD_WRAPPERSMALL_DESC"
         filter="int" />

          <field name="wrapperLarge" class="validate-numeric"
type="text" default="72"
         label="TPL_BEEZ2_FIELD_WRAPPERLARGE_LABEL"
         description="TPL_BEEZ2_FIELD_WRAPPERLARGE_DESC"
         filter="int" />

    <field name="logo" type="media"
        hide_default="1"
        label="TPL_BEEZ2_FIELD_LOGO_LABEL"
description="TPL_BEEZ2_FIELD_LOGO_DESC" />

      <field name="sitetitle" type="text" default=""
          label="TPL_BEEZ2_FIELD_SITETITLE_LABEL"
          description="TPL_BEEZ2_FIELD_SITETITLE_DESC"
          filter="string" />

      <field name="sitedescription" type="text" default=""
          label="TPL_BEEZ2_FIELD_DESCRIPTION_LABEL"
          description="TPL_BEEZ2_FIELD_DESCRIPTION_DESC"
          filter="string" />

      <field name="navposition" type="list" default="center"
          label="TPL_BEEZ2_FIELD_NAVPOSITION_LABEL"
          description="TPL_BEEZ2_FIELD_NAVPOSITION_DESC"
          filter="word">

          <option value="center">TPL_BEEZ2_OPTION_AFTER_CONTENT</option>
          <option value="left">TPL_BEEZ2_OPTION_BEFORE_CONTENT</option>
      </field>

      <field name="templatecolor" type="list" default="nature"
          label="TPL_BEEZ2_FIELD_TEMPLATECOLOR_LABEL"
          description="TPL_BEEZ2_FIELD_TEMPLATECOLOR_DESC"
          filter="word">

          <option value="nature">TPL_BEEZ2_OPTION_NATURE</option>
          <option value="personal">TPL_BEEZ2_OPTION_PERSONAL</option>
      </field>

    </fieldset>
  </fields>
</config>
```

Если сравнить содержимое XML-файла с экраном Advanced Options (Дополнительные параметры), приведенным на рис. 4.2, то можно заметить следующее:

- Каждая группа параметров определяется элементом разметки `fieldset` в XML-файле.
- Каждое поле параметра определяется отдельным элементом разметки `field` в XML-файле.

- Поведение каждого параметра определяется атрибутами, как поясняется вкратце в табл. 4.2.

Таблица 4.2. Атрибуты элементов разметки полей, определяющих параметры настройки шаблона

Атрибут	Описание
`name`	Имя, используемое для извлечения конкретного значения параметра
`class`	Класс вложенных таблиц стилей CSS для оформления поля при воспроизведении формы на странице. Например, класс `validate-numeric` запускает на выполнение специальный код JavaScript для отображения ошибок на странице при появлении нечислового символа
`type`	Имя файла исходного кода PHP и соответствующего класса, вызываемого при создании параметра. Так, если установить значение `text` в атрибуте `type`, то загружается файл `libraries/joomla/form/fields/text.php`, в котором определен класс `JformFieldText` языка PHP
`default`	Исходное значение параметра, отображаемое в форме при создании нового стиля оформления шаблона
`label`	Метка поля в форме. Как правило, она переводится, а следовательно, ее дескриптор обычно находится в файле языковой поддержки. Например, дескриптор `TPL_BEEZ2_FIELD_WRAPPERSMALL_LABEL` определен в файле `language/en-GB/en-GB.tpl_beez_20.ini`, где он переводится как `Wrapper Small (%)` (Мелкая оболочка (%)). Более подробно перевод текста элементов пользовательского интерфейса на разные языки мира рассматривается в главе 5
`description`	Описание поля в форме с отображением всплывающей подсказки при наведении курсора на само поле. Как и метка, описание поля, как правило, переводится
`filter`	Это фильтр. Если он установлен, то применяется к данным, вводимым в поле, при сохранении формы. Например, в поле `wrapperSmall` применяется фильтр `int`, а это означает, что в данном поле можно вводить только целочисленные значения. При наличии атрибута `filter` вызывается метод `clean()` класса `JFilterInput` из файла `libraries/joomla/filter/filterinput.php`. Перечень всех типов стандартных фильтров приведен в приложении Б

Элементы разметки `fieldset` и `field` были внедрены в версии Joomla 1.6 и заменили собой элементы разметки `params` и `param`, применявшиеся в версии 1.5.

Следует также заметить, что параметр `navposition` относится к типу `list`. Это означает, что пользователь может сделать выбор из раскрывающегося списка параметров шаблона. Каждый параметр в этом списке определен как отдельный элемент разметки `option`, а у каждого элемента разметки `option` имеется свой атрибут `value`, определяющий значение, передаваемое форме при выборе данного параметра из списка. Текст наименования элемента разметки `option` должен появляться в списке и, как правило, переводится, поэтому он представлен в виде дескриптора, определяемого для данного шаблона в файле языковой поддержки.

А теперь посмотрим, каким образом параметры сохраняются в базе данных веб-сайта Joomla. Если проанализировать содержимое таблицы `#__template_styles` базы данных, то в ней можно обнаружить строку со значением `Beez2 - Default` в столбце `title`. А если перейти к столбцу `params` в этой же строке, то в нем можно обнаружить следующее значение:

```
{"wrapperSmall":53,"wrapperLarge":72,"logo":"images\/joomla_black.gif",
"sitetitle":"Joomla!","sitedescription":"Open Source Content
Management","navposition":"left","templatecolor":"personal"}
```

Приведенный выше текст представлен в формате JSON (JavaScript Object Notation — представление объектов JavaScript). Начиная с версии 1.6 формат JSON используется для сохранения параметров настройки Joomla в базе данных. В рассматриваемом здесь примере сохранения параметров шаблона в формате JSON первым указывается имя параметра в кавычках, а после двоеточия — значение этого параметра. Каждый предыдущий параметр отделяется от последующего запятой, а вся запись параметров заключается в фигурные скобки.

С исчерпывающим описанием формата JSON можно ознакомиться по адресу `http://www.json.org`. В языке PHP имеется функция `json_decode()` для преобразования текста из формата JSON в объекты или массивы, а также функция `json_encode()` для преобразования массивов или объектов языка PHP в текст формата JSON.

Следует также заметить, что все параметры хранятся в единственном столбце `parameters` таблицы базы данных. Это означает, что новые параметры можно вводить, не добавляя новые столбцы в таблицу базы данных. Например, стиль оформления шаблона может содержать любое количество параметров с любыми именами, выбираемыми разработчиком шаблона. И такой подход к сохранению многих параметров в одном столбце таблицы базы данных принят повсеместно в Joomla, что в значительной степени упрощает процесс создания параметров. А недостаток такого подхода заключается в том, что пользоваться параметрами в запросах базы данных не так-то просто.

Стилевая рамка модуля

У элемента разметки `jdoc:include` для подстановки модулей (`type="module"`) имеется дополнительный атрибут `style`. Например, в файле `index.php` шаблона `atomic` можно обнаружить следующую строку кода:

```
<jdoc:include type="modules" name="atomic-topmenu" style="container" />
```

Стиль `container` указывает Joomla искать функцию `modChrome_container()` в одном из двух файлов. Стандартные стили находятся в файле `templates/system/html/modules.php`. А дополнительные стили определены как функции в файле `modules.php`, находящемся в папке `html` данного шаблона. Например, для шаблона `atomic` три дополнительных стиля оформления модулей определены в файле `templates/atomic/html/modules.php`, как показано в табл. 4.3.

Таблица 4.3. Стили оформления модулей в шаблоне `atomic`

Наименование стиля	Наименование функции
`container`	`modChrome_container`
`bottommodule`	`modChrome_bottommodule`
`Sidebar`	`modChrome_sidebar`

Следует иметь в виду, что в файлах `modules.php` объявляются только функции, но не классы PHP. Это дает некоторые преимущества при добавлении новых стилей оформления модулей. В частности, соблюдать условные обозначения имен классов не нужно,

поскольку отсутствуют сами имена классов. Нужно лишь найти файл исходного кода PHP и включить его в проект.

Рассмотрим в качестве примера функцию `modChrome_container()` из файла `modules.php` для шаблона `atomic`. Она определяет стиль `container`, а ее код приведен в листинге 4.5.

Листинг 4.5. Функция, определяющая стиль `container` из шаблона `atomic`

```
function modChrome_container($module, &$params, &$attribs)
{
  if (!empty ($module->content)) : ?>
    <div class="container">
        <?php echo $module->content; ?>
    </div>
  <?php endif;
}
```

Этой функции передаются три аргумента. В качестве первого аргумента, `$module`, передается объект, содержащий поля, приведенные в табл. 4.4; в качестве второго аргумента, `JRegistry`, — объект с параметрами настройки данного модуля, что удобно для проверки значения любого параметра в функции стилевого оформления; а в качестве третьего аргумента — массив атрибутов из элемента разметки `jdoc:include` (например, `name`, `style` и т.д.). Следует заметить, что в дескрипторе `jdoc:include` можно ввести любые атрибуты, которые окажутся доступными в функции типа `modChrome()`. И это может в дальнейшем облегчить специальную настройку внешнего вида модулей.

Таблица 4.4. Поля объекта `$module`, передаваемого функциям типа `modChrome()`

Наименование поля	Описание
`id`	Идентификационный номер модуля (в таблице `#__modules`)
`title`	Заголовок модуля, введенного в диспетчере модулей (например, "`Login Form`" — форма регистрации)
`module`	Имя папки с модулем (например, `mod_login`)
`position`	Место расположения модуля в шаблоне
`content`	Текст содержимого модуля (HTML-текст, отображаемый на странице внутри стилевой рамки модуля, реализуемой соответствующей функцией)
`showtitle`	Определяет, следует ли отображать заголовок модуля
`params`	Параметры модуля в формате JSON
`menuid`	Идентификационный номер пункта меню, при выборе которого воспроизводится модуль
`user`	Не используется
`name`	Имя модуля (например, `login`)
`style`	Стиль оформления модуля, выбираемый из элемента разметки `jdoc:include`

Допустим, стилевая рамка модуля имеет обрамление и при выделении места в шаблоне для расположения данного модуля требуется задать ширину линии обрамления. Это можно, в частности, реализовать, добавив сначала атрибут `border_width` в элемент разметки

jdoc:include, а затем передав его в качестве аргумента функции типа modChrome(), чтобы задать ширину линии обрамления. В зависимости от значения атрибута border_width для этого можно выбрать подходящий класс вложенных таблиц стилей CSS.

Рассмотрим тело самой функции modChrome_container(). В первой строке кода этой функции проверяется наличие любого содержимого для данного модуля. Если такое содержимое отсутствует, то в модуле ничего не отображается. А если оно присутствует, то создается элемент разметки div и в нем выводится содержимое модуля.

Более сложный пример стилевого оформления модуля можно найти в шаблоне beez_20 (см. файл templates/beez_20/html/modules.php). Но в любом случае конкретный результат выполнения модуля содержится в переменной $module->content. А поскольку функция стилевого оформления модуля из файла modules.php предоставляется как составная часть шаблона, то в него можно добавить сколько угодно стилей, чтобы специально настроить внешний вид модулей на веб-сайте.

Копирование шаблона

Все файлы замещений компоновок размещаются в папке templates/<имя_шаблона>/html. Так, если шаблон называется my_beautiful_template, то все файлы замещений компоновок будут располагаться в папке templates/my_beautiful_template/html. Это означает, что файлы замещений благополучно сохраняются в папке отдельного шаблона. Но такое "благополучное" сохранение возможно лишь в том случае, если шаблон не является базовым.

Базовые шаблоны, входящие в состав Joomla, могут быть изменены при переходе к новой версии Joomla. И это не вызывает никаких сложностей, если шаблон используется без всяких изменений в его файлах. Но если в шаблоне производятся замены, то в конечном итоге одни его файлы могут быть изменены как старые, а другие добавлены как новые.

Обезопасить файлы от замены и обновления при переходе к новой версии Joomla совсем не трудно. Для этого достаточно воспользоваться шаблоном, не включенным в основной вариант распространения Joomla. Ради простоты изложения материала в этой книге будет использоваться копия beez_20_copy исходного шаблона beez_20. Получить копию этого шаблона можно двумя способами.

- Создать собственную копию шаблона beez_20_copy, следуя приведенным ниже инструкциям.
- Загрузить архивный файл beez_20_copy.zip с веб-сайта, посвященного этой книге (http://joomlaprogrammingbook.com/downloads.html), а затем произвести установку по команде Extension Manager⇨Install (Диспетчер расширений⇨Установить).

Сделать копию шаблона относительно легко и даже полезно, чтобы знать, как это вообще делается. Для этого выполните следующие действия.

1. Скопируйте папку шаблона в новую папку. С этой целью воспользуйтесь функцией копирования папок на уровне операционной системы, чтобы переместить папку templates/beez_20 (со всеми входящими в нее подпапками и файлами) в новую папку templates/beez_20_copy. Непременно убедитесь в том, что все файлы исходной папки правильно скопированы в целевую папку.

2. Измените имена файлов в XML-файле шаблона `beez_20_copy`. С этой целью откройте XML-файл `beez_20_copy/templateDetails.xml` для редактирования. Само же редактирование состоит лишь в глобальном поиске и замене текста `beez_20` на текст `beez_20_copy`. Таких замен должно быть три: в элементе `name` и двух файлах языковой поддержки в элементах `language`.
3. Переименуйте файлы языковой поддержки. В п.2 данной процедуры имена файлов были изменены в XML-файле. А теперь нужно переименовать их на уровне файловой системы следующим образом:
   - `en-GB.tpl_beez_20.ini` на `en-GB.tpl_beez_20_copy.ini`
   - `en-GB.tpl_beez_20.sys.ini` на `en-GB.tpl_beez_20_copy.sys.ini`
4. Скопируйте оба эти файла из папки `templates/beez_20_copy/language/en-GB/` в папку `language/en-GB/`, вновь воспользовавшись функцией копирования на уровне операционной системы.
5. Найдите и установите копию шаблона в качестве нового расширения. В версии 1.6 внедрено новое замечательное средство обнаружения программных компонентов Joomla — Discover. Оно предоставляет альтернативный способ установки новых расширений. При установке расширения происходит следующее.
   - Файлы расширения копируются в нужные папки текущей установки Joomla.
   - База данных Joomla обновляется информацией о новом расширении.

С помощью средства обнаружения Discover можно вручную скопировать файлы в папки, а затем дать Joomla команду найти новые расширения. Как только новое расширение будет обнаружено, его можно установить. Благодаря этому в базе данных Joomla создаются правильные записи. Воспользуемся этим средством, чтобы установить новый шаблон. Если перейти по команде Extension Manager⇨Discover (Диспетчер расширений⇨Обнаружить) и выбрать пиктограмму Discover на панели инструментов, то Joomla найдет новый шаблон, как показано на рис. 4.3.

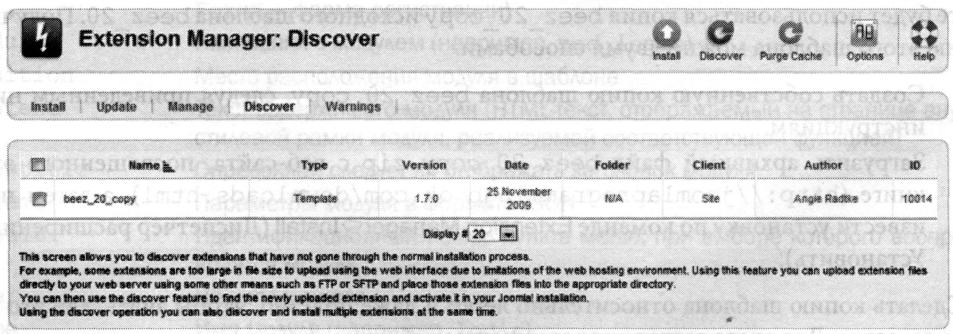


Рис. 4.3. Обнаружение нового расширения для его установки

Теперь можно установить новый шаблон, щелкнув сначала на поле его выбора, а затем на пиктограмме Install непосредственно на панели инструментов. По завершении установки нового шаблона будет выдано сообщение "`Discover Install Successful`" (Обнаруженное установлено успешно).

Далее шаблон `beez_20_copy` требуется сделать выбираемым для веб-сайта по умолчанию. Для этого перейдите по команде Extensions⇨Template Manager: Styles (Расширения⇨Диспетчер шаблонов: Стили), выберите новый шаблон и щелкните на кнопке Make Default (Сделать выбираемым по умолчанию), как показано на рис. 4.4.

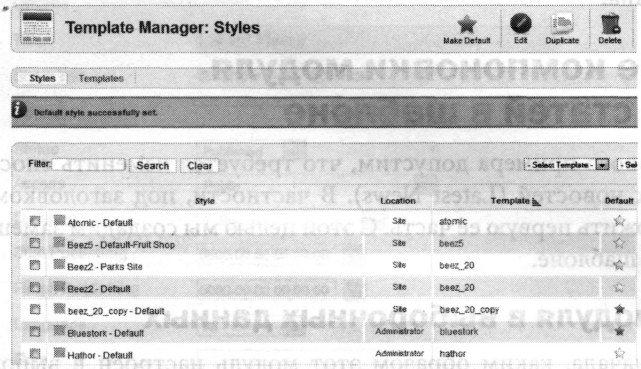


Рис. 4.4. Новый шаблон, выбираемый по умолчанию

И наконец, если требуется, чтобы новый шаблон выглядел как исходный, нужно изменить в нем параметры нового стиля таким образом, чтобы привести его в соответствие со старым стилем оформления. Это делается в экранной форме `Edit Style` (Правка стиля), открываемой щелчком на имени шаблона (в данном случае — `beez_20_copy - Default`), как показано на рис. 4.5.

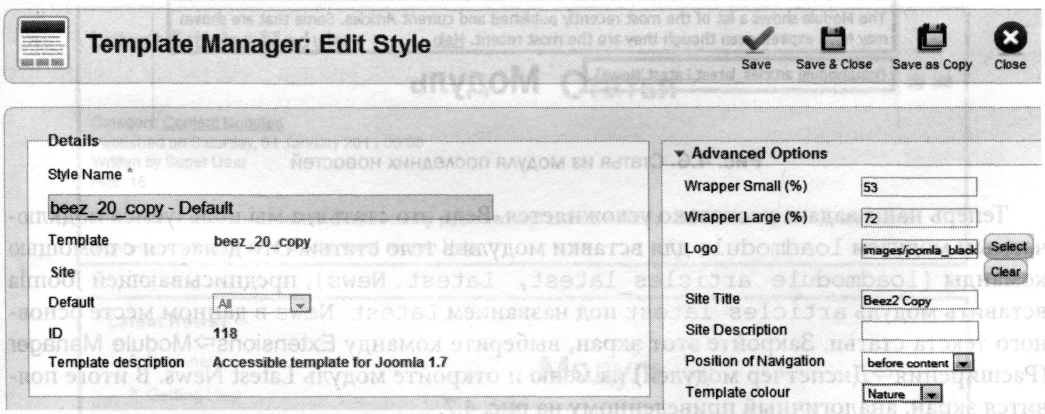


Рис. 4.5. Правка стиля оформления нового шаблона

Поля в открывшейся форме устанавливаются следующим образом:

- Logo (Логотип): `joomla_black.gif`
- Position of navigation (Местоположение при навигации): `Before content` (До содержимого)
- Template colour (Цвет шаблона): `Personal` (Индивидуально выбираемый)

При этих настройках параметров стиля новый шаблон будет выглядеть точно так же, как и исходный шаблон `beez_20`. И теперь мы можем приступить к созданию замещений компоновок. Созданные нами файлы замещений будут храниться в нашем собственном шаблоне, а следовательно, внесенные нами изменения не будут потеряны при переходе к новой версии Joomla.

Замещение компоновки модуля последних статей в шаблоне

В качестве первого примера допустим, что требуется изменить способ отображения модуля последних новостей (Latest News). В частности, под заголовком каждой статьи требуется расположить первую ее часть. С этой целью мы создадим замещение компоновки этого модуля в шаблоне.

Настройка модуля в выборочных данных

Рассмотрим сначала, каким образом этот модуль настроен в выборочных данных. С этой целью выберите команду Content⇨Article Manager (Содержимое⇨Диспетчер статей) из меню в административной части Joomla и откройте статью из модуля Latest News для редактирования. В итоге появится экран, аналогичный приведенному на рис. 4.6.

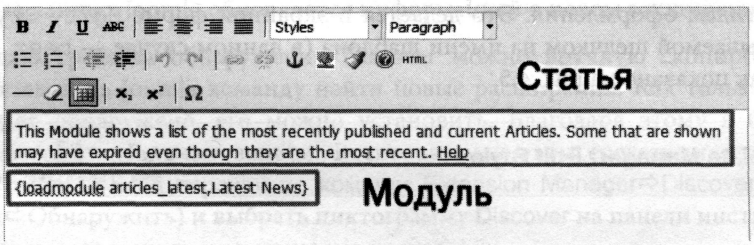


Рис. 4.6. Статья из модуля последних новостей

Теперь наша задача несколько усложняется. Ведь это статья, а мы пользуемся подключаемым модулем `loadmodule` для вставки модуля в тело статьи. Это делается с помощью команды `{loadmodule articles_latest, Latest News}`, предписывающей Joomla вставить модуль `articles_latest` под названием `Latest News` в данном месте основного текста статьи. Закройте этот экран, выберите команду Extensions⇨Module Manager (Расширения⇨Диспетчер модулей) из меню и откройте модуль Latest News. В итоге появится экран, аналогичный приведенному на рис. 4.7.

Далее выберите команду Using Joomla!⇨Using Extensions⇨Modules⇨Content Modules⇨Latest Articles (Применение Joomla!⇨Расширения⇨Модули⇨Модули содержимого⇨Последние статьи) из меню в пользовательской части Joomla. В итоге появится экран со статьей из модуля Latest Articles, как показано на рис. 4.8. В верхней части этого экрана находится первая часть статьи (вплоть до дескриптора `{loadmodule}`), а в нижней его части — конкретный модуль Latest News. Как видите, это простой перечень заголовков статей в виде ссылок на сами статьи.

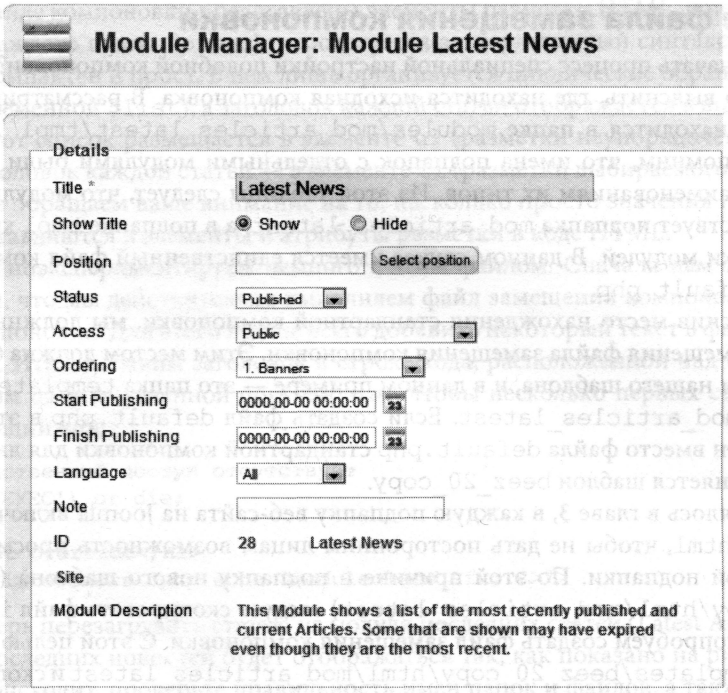


Рис. 4.7. Модуль последних новостей

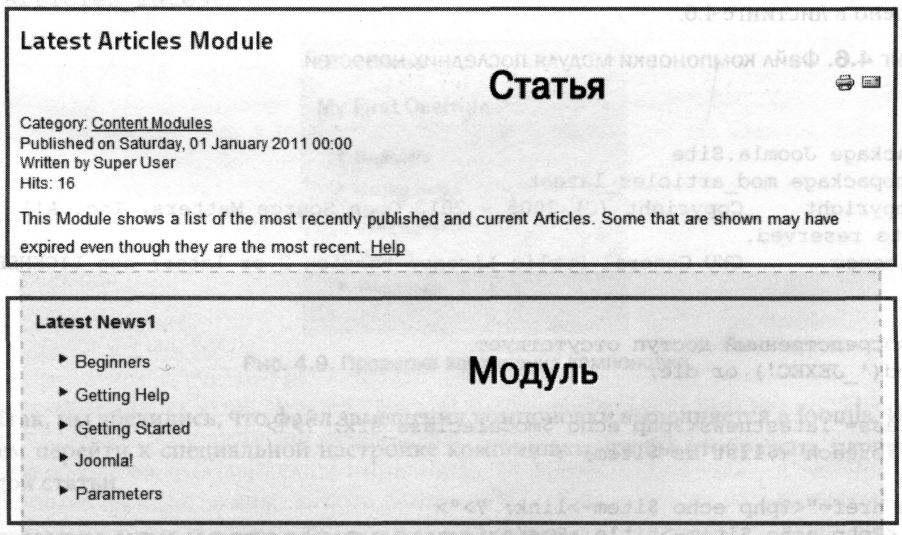


Рис. 4.8. Статья из модуля последних статей

Создание файла замещения компоновки

С чего же начать процесс специальной настройки подобной компоновки? Прежде всего нам нужно выяснить, где находится исходная компоновка. В рассматриваемом здесь примере она находится в папке modules/mod_articles_latest/tmpl. А откуда это известно? Напомним, что имена подпапок с отдельными модулями были приведены в табл. 3.2 по наименованиям их типов. Из этой таблицы следует, что модулю типа Latest News соответствует подпапка mod_articles_latest, а в подпапке tmpl хранятся файлы компоновки модулей. В данном случае имеется единственный файл компоновки под названием default.php.

Итак, выяснив место нахождения стандартной компоновки, мы должны определить место для размещения файла замещения компоновки. Этим местом должна быть подпапка html папки нашего шаблона, и в данном примере — это папка templates/beez_20_copy/html/mod_articles_latest. Если создать файл default.php в этой папке, он будет загружен вместо файла default.php стандартной компоновки для любой страницы, где применяется шаблон beez_20_copy.

Как пояснялось в главе 3, в каждую подпапку веб-сайта на Joomla включается пустой файл index.html, чтобы не дать посторонним лицам возможность просматривать содержимое этой подпапки. По этой причине в подпапку нового шаблона (templates/beez_20_copy/html/mod_articles_latest) следует скопировать файл index.html.

И теперь попробуем создать файл замещения компоновки. С этой целью создайте новую папку templates/beez_20_copy/html/mod_articles_latest и скопируйте файл modules/mod_articles_latest/tmpl/default.php на место файла templates/beez_20_copy/html/mod_articles_latest/default.php. Содержимое этого файла приведено в листинге 4.6.

Листинг 4.6. Файл компоновки модуля последних новостей

```
<?php
/**
 * @package Joomla.Site
 * @subpackage mod_articles_latest
 * @copyright     Copyright (C) 2005 - 2012 Open Source Matters, Inc. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

// непосредственный доступ отсутствует
defined('_JEXEC') or die;
?>
<ul class="latestnews<?php echo $moduleclass_sfx; ?>">
<?php foreach ($list as $item) : ?>
  <li>
    <a href="<?php echo $item->link; ?>">
      <?php echo $item->title; ?></a>
  </li>
<?php endforeach; ?>
</ul>
```

В этом файле компоновки представлены элементы разметки HTML, и поэтому для организации каждого цикла `foreach` используется альтернативный синтаксис языка PHP. Этот файл компактен и прост. В нем лишь организуется циклическое обращение к списку статей (в переменной `$list`), а заголовок каждой статьи отображается в виде ссылочного элемента. Этот список размещается в элементе `ul` (разметки неупорядоченного списка), тогда как заголовок каждой статьи — в элементе `li` (разметки выбираемого элемента списка). Еще раз обращаем ваше внимание на то, насколько просто значения переменных из кода PHP вставляются в элементы и атрибуты разметки в коде HTML.

А теперь поэкспериментируем немного с этим файлом. Сначала нам требуется убедиться в том, что мы действительно выполняем файл замещения компоновки, а не стандартной компоновки. Для этого проще всего добавить некоторый текст в файл замещения компоновки. Итак, добавим заголовок в строке кода, расположенной над открывающим дескриптором `ul` и выделенной полужирным, чтобы несколько первых строк кода приняли следующий вид:

```
// непосредственный доступ отсутствует
defined('_JEXEC') or die;
?>
<h1>My First Override</h1>
<ul class="latestnews<?php echo $moduleclass_sfx; ?>">
```

Если теперь перезагрузить статью из модуля последних статей (Latest Articles Module), то модуль последних новостей будет отображаться так, как показано на рис. 4.9. Если же этого не происходит, проверьте правильность имен папок и файлов, а также наличие отредактированной версии файла компоновки в папке `templates/beez_20_copy/html/mod_articles_latest`.

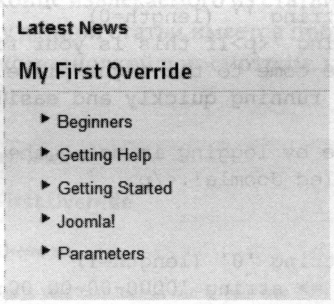


Рис. 4.9. Проверка замещения компоновки

Итак, мы убедились, что файл замещения компоновки выполняется в Joomla. И теперь можем перейти к специальной настройке компоновки, чтобы отображать первую часть каждой статьи.

Специальная настройка компоновки

Если еще раз проанализировать содержимое файла `default.php`, то можно заметить, что заголовок статьи и ссылка на нее извлекаются из переменной `$item`. Циклическое обращение к элементам массива организуется с помощью оператора `foreach` языка PHP,

а следовательно, переменная `$list` содержит массив объектов, где каждый объект доступен из переменной `$item`.

Мы знаем также, что у объекта из переменной `$item` имеются, по крайней мере, два поля: `link` и `title`, поскольку они используются в компоновке. Но из данного файла мы не можем выяснить, какая еще информация в ней имеется. Правда, это нетрудно будет сделать, если добавить в файл компоновки команду `var_dump($item)`, расположив ее в цикле `foreach`. В итоге все содержимое переменной `$item` будет выведено на экран, и мы узнаем, что еще в ней хранится. Ниже показано, каким образом выглядит видоизмененный файл компоновки, где упомянутая выше команда выделена полужирным.

```
<?php foreach ($list as $item) : ?>
  <li>
    <a href="<?php echo $item->link; ?>">
      <?php echo $item->title; ?></a>
      <?php var_dump($item);?>
  </li>
<?php endforeach; ?>
```

Если теперь перезагрузить статью из модуля последних статей (Latest Articles Module), появится длинный перечень содержимого, выведенного из переменной `$item` с помощью функции `var_dump()`. Частичное его представление приведено в листинге 4.7.

Листинг 4.7. Содержимое переменной `$item`, выводимое с помощью функции `var_dump()`

```
Beginners
object(stdClass)[539]
  public 'id' => string '8' (length=1)
  public 'title' => string 'Beginners' (length=9)
  public 'alias' => string 'beginners' (length=9)
  public 'title_alias' => string '' (length=0)
  public 'introtext' => string '<p>If this is your first Joomla! site or
your first website, you have come to the right place. Joomla! will help
you get your website up and running quickly and easily.</p>

<p>Start off using your site by logging in using the administrator account
you created when you installed Joomla!.</p>

' (length=293)
  public 'checked_out' => string '0' (length=1)
  public 'checked_out_time' => string '0000-00-00 00:00:00' (length=19)
  public 'catid' => string '19' (length=2)
```

Если воспользоваться отладчиком кода PHP, то можно получить ту же самую информацию, вставив точку прерывания там, где была введена функция `var_dump()`, а затем проанализировать содержимое переменных средствами отладчика. Из выводимой информации выясняется, что переменная `$item` содержит объект типа `stdClass`, встроенный в PHP и действующий аналогично массиву. Каждое поле этого объекта доступно по записи `<объектная переменная>-><имя поля>`. Таким образом, в стандартной компоновке мы обнаруживаем код `$item->title`, предоставляющий доступ к полю `title` внутри объекта из переменной `$item`. Кроме того, из выводимой информации выясняется тип каждого поля (строковый, целочисленный и т.д.), его значение и длина.

Мы обнаруживаем большое количество полей, доступных для статьи из данной компоновки. Но для рассматриваемого здесь примера нам требуется лишь одно поле — `introtext`. В этом поле хранится часть текста статьи вплоть до прерывающей ссылки `Read More` (Читать далее). А в отсутствие прерывающей ссылки `Read More` в этом поле хранится весь текст статьи.

Напомним, что наша задача — расположить первую часть каждой статьи под ее заголовком. Для этого можно было бы, например, воспользоваться функцией `substr()` языка PHP, которая позволяет извлечь часть символьной строки из заданного в ней места. Допустим, что нам требуется отобразить первых 50 символов из каждой статьи. С этой целью мы могли бы видоизменить компоновку так, как выделено ниже полужирным.

```
<li>
  <a href="<?php echo $item->link; ?>">
    <?php echo $item->title; ?></a>
    <?php echo substr($item->introtext, 0, 50) . '...'; ?>
</li>
```

Функция `substr()` принимает три аргумента, указываемых в круглых скобках. В качестве первого аргумента указывается исходный текст, в качестве второго — место, с которого следует начинать извлечение фрагмента этого текста, и в качестве третьего аргумента — длину извлекаемого фрагмента. В данном случае нам требуется извлечь фрагмент текста длиной в **50** символов с самого начала статьи, для чего в качестве второго аргумента указывается **0**, а не **1**.

Для начинающих осваивать язык PHP заметим, что оператор точки (`"."`) называется *сцеплением*. Этим замысловатым термином обозначается обычное соединение двух текстовых или символьных строк в одно целое. Таким образом, оператор сцепления `. '...'` лишь добавляет многоточие в конце извлекаемого из статьи начального фрагмента текста, давая пользователю знать, что у данной статьи имеется продолжение. А теперь перезагрузим страницу с поправленной компоновкой и посмотрим, как она выглядит после внесенных нами корректив (рис. 4.10).

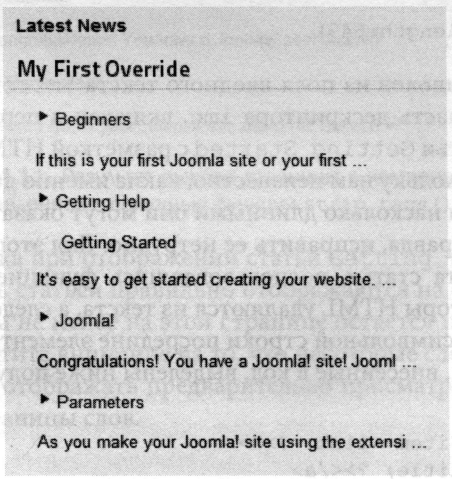


Рис. 4.10. Замещение вводного текста

Устранение ошибки с помощью функции strip_tags()

На первый взгляд страница выглядит именно так, как нам требуется: первая часть каждой статьи отображается под ее заголовком, после чего следует многоточие, указывающее на продолжение статьи. Но если присмотреться к данной странице внимательнее, то можно обнаружить ошибку. Третьей по списку должна следовать статья Getting Started (Начало работы), но она отображается под статьей Getting Help (Получение справки). Что же произошло?

Для того чтобы ответить на этот вопрос, проанализируем текст статьи Getting Help. Если вернуться к выводу содержимого переменной $item с помощью функции var_dump() и посмотреть содержимое поля introtext для статьи Getting Help, то в нем обнаружится длинный дескриптор img, как показано в листинге 4.8.

Листинг 4.8. Вводный текст статьи Getting Help

```
Getting Help
object(stdClass)[551]
  public 'id' => string '21' (length=2)
  public 'title' => string 'Getting Help' (length=12)
  public 'alias' => string 'getting-help' (length=12)
  public 'title_alias' => string '' (length=0)
public 'introtext' => string '<p><img class="image-left"
↳src="administrator/templates/hathor/images/header/icon-48-
↳help_header.png" border="0" /> There are lots of places you can get help
↳with Joomla!. In many places in your site administrator you will see the
↳help icon. Click on this for more information about the options and
↳functions of items on your screen. Other places to get help are:</p>

<ul>

<li><a href="http://forum.joomla.org">Support Forums</a></li>

<li><a href="http://docs.joomla.org">Documentation</a></li>

<li><a href="' . . . (length=649)
```

Выбрав первые 50 символов из поля вводного текста introtext, мы, как оказалось, получили лишь первую часть дескриптора img, включая и первую часть атрибута src. Таким образом, текст статьи Getting Started с разметкой HTML был неверно вставлен в дескрипторе img. А поскольку нам неизвестно, какие именно дескрипторы HTML могут присутствовать в статье и насколько длинными они могут оказаться, то и возникает упомянутая выше ошибка. Правда, исправить ее нетрудно. Для этого достаточно исключить разметку HTML из текста статьи, воспользовавшись функцией strip_tags() языка PHP. В итоге все дескрипторы HTML удаляются из текста, а следовательно, не нужно беспокоиться о прерывании символьной строки посредине элемента разметки HTML. Соответствующие коррективы, внесенные в код, выделены ниже полужирным.

```
<li>
  <a href="<?php echo $item->link; ?>">
    <?php echo $item->title; ?></a>
    <p>
```

```
        <?php echo substr(strip_tags($item->introtext), 0, 50) . '...'; ?>
    </p>
</li>
```

Начинающим осваивать программирование строка, выделенная полужирным в приведенном выше фрагменте кода, может показаться не совсем понятной, поскольку в ней результат выполнения одной функции помещается в вызове другой функции (в данном случае — `substr()`). На самом же деле это означает, что код PHP получает команду сначала исключить дескрипторы из вводного текста в поле `$item->introtext`, а затем передать полученный результат функции `substr()`, которая извлекает из него первые 50 символов и добавляет к ним многоточие, а конечный результат передается браузеру по команде `echo`.

Обратите внимание на то, что предварительно просматриваемый текст статьи размещается в элементе разметки `p`. И делается это потому, что в поле `introtext` отсутствует какая-либо разметка HTML. Благодаря размещению текста в элементе разметки `p` гарантируется, что он достоверно размечен и отформатирован как абзац HTML-документа. Результат вывода страницы с исправленной компоновкой приведен на рис. 4.11.

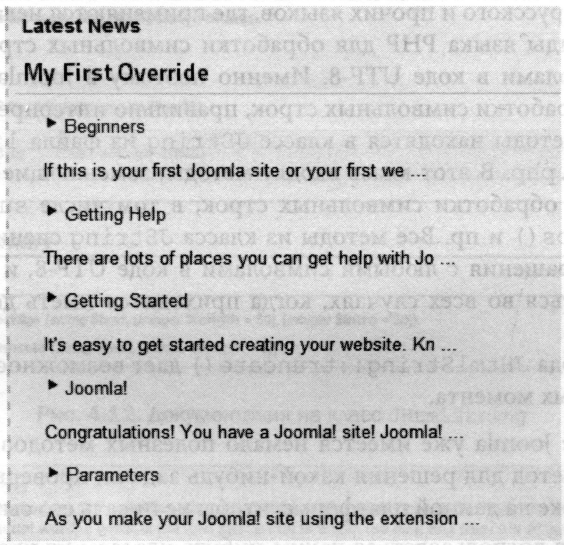


Рис. 4.11. Результат вывода страницы с компоновкой, исправленной с помощью функции `strip_tags()`

Таким образом, ошибка при отображении статьи `Getting Started` на странице исправлена. Теперь все пять статьей правильно отображаются на странице с 50 символами их начального текста. Тем не менее на этой странице остается не устраненным еще один мелкий недостаток. Обратите внимание на то, что некоторые слова обрываются посредине. Правильнее было бы отображать предварительно просматриваемый текст статей, не нарушая естественные границы слов.

Применение метода `truncate()` из класса `JHtmlString`

Для устранения этого недостатка можно было бы написать собственный метод, воспользовавшись, например, функцией `strrpos()` языка PHP, чтобы обнаружить последний пробел в предварительно просматриваемом (или вводном) тексте каждой статьи и отобразить его только до этого пробела. Но в Joomla уже имеется метод, предназначенный именно для этой цели. Он называется `truncate()` и находится в файле `libraries/joomla/html/html/string.php` библиотеки Joomla.

Если проанализировать код этого метода, то можно заметить, что в нем применяется метод `JString::substr()` вместо стандартной для языка PHP функции `substr()`, как показано в приведенной ниже строке кода.

```
$tmp = JString::substr($text, 0, $length);
```

Почему выбран именно такой подход? Ответ следует искать к международной области применения Joomla. Для поддержки всех основных языков мира приходится правильно обращаться с кодировкой UTF-8. Этот стандартный набор символов позволяет отображать все символы алфавита любого языка мира, в том числе китайского, греческого, корейского, арабского, русского и прочих языков, где применяются нелатинские символы.

Некоторые команды языка PHP для обработки символьных строк не совсем верно обращаются с символами в коде UTF-8. Именно поэтому в Joomla применяются собственные методы обработки символьных строк, правильно интерпретирующие символы в коде UTF-8. Эти методы находятся в классе `JString` из файла `libraries/joomla/utilities/string.php`. В этот класс входят методы, заменяющие стандартные функции языка PHP для обработки символьных строк, в том числе `substr()`, `strlen()`, `str_split()`, `strpos()` и пр. Все методы из класса `JString` специально разработаны для правильного обращения с любыми символами в коде UTF-8, и поэтому ими рекомендуется пользоваться во всех случаях, когда приходится иметь дело с нелатинскими символами.

Применение метода `JHtmlString::truncate()` дает возможность обратить внимание на еще два важных момента.

1. На платформе Joomla уже имеется немало полезных методов. Поэтому если вам потребуется метод для решения какой-нибудь задачи, проверьте сначала, присутствует ли он уже на данной платформе, чтобы не писать его сызнова.
2. Если требуется воспользоваться каким-нибудь классом из платформы Joomla, его следует сделать доступным для компоновки. Ниже будет показано, как это можно сделать разными способами.

В состав библиотеки (или платформы) Joomla входит немало полезных методов, решающих типичные задачи веб-разработки. Так, в папке `html` находится библиотека с методами, особенно пригодными для этой цели. Перечень удобных методов, пригодных для разметки HTML-документов, приведен в приложении B. Таким образом, решая задачу, требующую программирования, следует сначала выяснить, имеется ли готовое ее решение на платформе Joomla. В связи с этим целесообразно сделать небольшое отступление, чтобы ответить на следующий вопрос: как найти на платформе Joomla подходящие методы для решения конкретных задач веб-разработки?

Для такого поиска имеется несколько источников информации. Так, в приложении В представлён ряд методов, особенно полезных для веб-разработки на платформе Joomla. Кроме того, можно перейти на страницу `Code Summary` (Сводка кода) веб-сайта разработчиков Joomla по адресу `http://developer.joomla.org/code.html` и щелкнуть на ссылке `API Documentation` (Документация на прикладной программный интерфейс), чтобы перейти далее на сайт `http://api.joomla.org`, где представлены все классы последней версии платформы Joomla. Так, если выбрать сначала пакет `Joomla-Platform` из раскрывающегося списка слева вверху, а затем элемент `HTML⇨Classes⇨JHtmlString` из расположенного ниже древовидного списка, на странице появятся сведения о классе `JHtmlString` (рис. 4.12). А если прокрутить страницу вниз, то можно обнаружить сведения о методе `truncate()`, как показано на рис. 4.13.

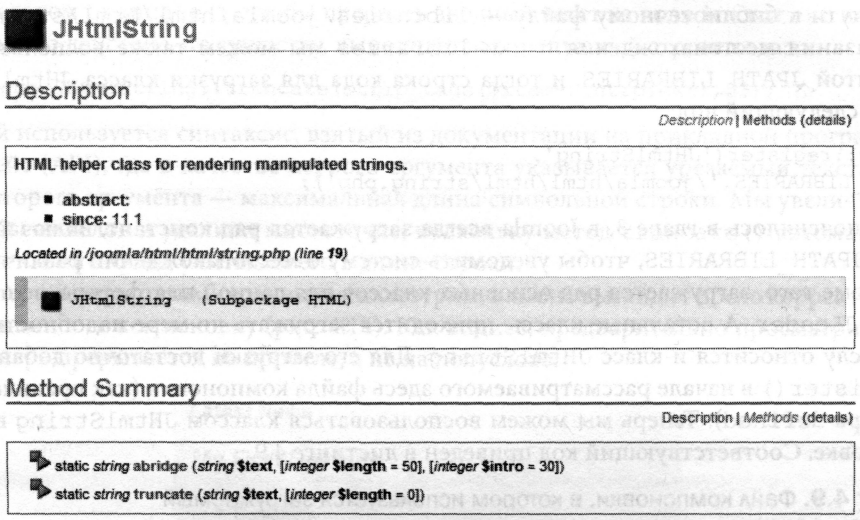


Рис. 4.12. Документация на класс `JHtmlString`

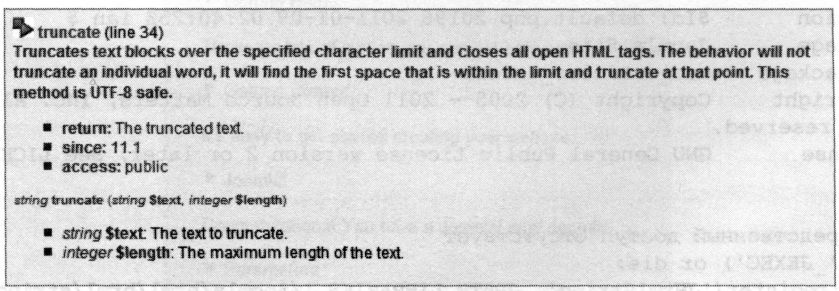


Рис. 4.13. Сведения о методе `truncate()` из класса `JHtmlString`

Пытаясь выяснить, существует ли нужный вам метод на платформе Joomla, вы можете обратиться с вопросом по одному из каналов общения с различными разработчиками Joomla. Еще раз напоминаем, что если вам требуется метод для решения какой-нибудь

задачи, попробуйте сначала найти его на платформе Joomla, а уж затем приступать к написанию собственного метода, если ничего подходящего обнаружить не удастся.

А теперь покажем, как добавить библиотечный файл в текущую компоновку. На рис. 4.12 обратите внимание на строку `Located in...` (Находится в...) под заголовком `Description` (Описание). В ней указано местонахождение нужного нам библиотечного файла, чтобы включить его в текущую компоновку. В данном случае файл находится по пути `joomla/html/html/string.php`. Напомним, что все классы платформы Joomla находятся в папке `libraries`, и поэтому полный путь к искомому файлу следующий: `libraries/joomla/html/html/string.php`.

Для загрузки класса `JHtmlString` воспользуемся методом `JLoader::register()`. Метод `register()` принимает два аргумента: имя класса и полный путь к библиотечному файлу. В данном случае в качестве имени класса указывается `JHtmlString`, а в качестве пути к библиотечному файлу — `libraries/joomla/html/html/string.php`. Для указания местонахождения папки `libraries` мы можем также воспользоваться константой `JPATH_LIBRARIES`, и тогда строка кода для загрузки класса `JHtmlString` примет следующий вид:

```
JLoader::register('JHtmlString',
↳JPATH_LIBRARIES.'/joomla/html/html/string.php');
```

Как пояснялось в главе 3, в Joomla всегда загружается ряд констант, включая и константу `JPATH_LIBRARIES`, чтобы уведомить систему о местонахождении различных папок. Кроме того, загружается ряд основных классов для данной платформы, в том числе и класс `JLoader`. А остальные классы приходится загружать по мере надобности в них. К их числу относится и класс `JHtmlString`. Для его загрузки достаточно добавить метод `register()` в начале рассматриваемого здесь файла компоновки (как правило, после оператора `defined`). Теперь мы можем воспользоваться классом `JHtmlString` в нашей компоновке. Соответствующий код приведен в листинге 4.9.

Листинг 4.9. Файл компоновки, в котором используется загружаемый метод `JHtmlString::truncate()`

```
<?php
/**
 * @version       $Id: default.php 20196 2011-01-09 02:40:25Z ian $
 * @package       Joomla.Site
 * @subpackage    mod_articles_latest
 * @copyright     Copyright (C) 2005 - 2011 Open Source Matters, Inc. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

// непосредственный доступ отсутствует
defined('_JEXEC') or die;
JLoader::register('JHtmlString', JPATH_LIBRARIES.'/joomla/html/html/string.php');
?>
<h1>My First Override</h1>
<ul class="latestnews<?php echo $moduleclass_sfx; ?>">
<?php foreach ($list as $item) : ?>
  <li>
```

```
    <a href="<?php echo $item->link; ?>">
      <?php echo $item->title; ?></a>
      <p>
        <?php echo JHtmlString::truncate(strip_tags($item->introtext),53); ?>
      </p>
  </li>
<?php endforeach; ?>
</ul>
```

Если попробовать воспользоваться классом `JHtmlString` без предварительной загрузки с помощью метода `register()`, то появится сообщение об ошибке в коде PHP, уведомляющее о том, что класс `JHtmlString` не удалось найти. А теперь, когда у нас имеется непосредственный доступ к методам из класса `JHtmlString`, можем воспользоваться его методом `truncate()`, чтобы прервать вводный текст из поля `introtext` на границе слова. Это действие, по существу, выполняется в следующей строке кода:

```
<?php echo JHtmlString::truncate(strip_tags($item->introtext),53); ?>
```

В ней используется синтаксис, взятый из документации на прикладной программный интерфейс (API), где в качестве первого аргумента указывается урезаемый текст, а в качестве второго аргумента — максимальная длина символьной строки. Мы увеличили ее с **50** до **53** символов и удалили многоточие, поскольку метод `truncate()` автоматически добавляет многоточие, учитывая его в длине строки.

А теперь перезагрузим страницу и посмотрим, как она выглядит после корректив, внесенных нами в ее компоновку (рис. 4.14). Как видите, предварительно просматриваемый текст теперь прерывается на пробеле, а не на полуслове.

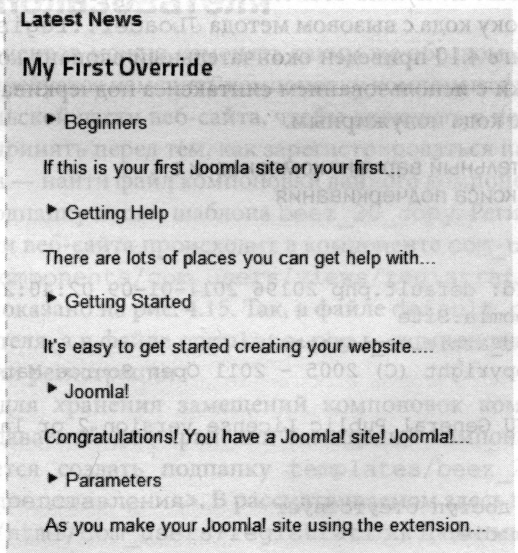


Рис. 4.14. Результат компоновки страницы с использованием метода `truncate()`

Применения синтаксиса подчеркивания

Итак, мы почти у цели нашего изыскания. В завершение рассматриваемого здесь примера рассмотрим один весьма изящный прием программирования. Если попытаться найти в кодовой базе Joomla места, где применяется метод `JHtmlString::truncate()`, то мы не обнаружим ничего подобного, а вместо этого — строки кода, аналогичные следующей:

```
<?php echo JHTML::_('string.truncate', $item->introtext,
↳$params->get('introtext_limit')); ?>
```

В этой строке кода применяется другой способ вызова того же самого метода `truncate()`, и делается это с помощью метода с синтаксисом подчеркивания (`_`) из класса `JHTML`. (Следует иметь в виду, что в именах классов языка PHP регистр не учитывается, и поэтому имена `JHTML` и `JHtml` обозначают один и тот же класс. Но нам приходится указывать имя `JHtml`, чтобы придерживаться принятых в Joomla условных обозначений имен, тогда как для самого кода PHP это не имеет никакого значения.)

Метод `JHtml::_()` выполняет очень необычную, но замечательную функцию, интерпретируя первый свой аргумент следующим образом. Первая половина этого аргумента (`string`) обозначает часть имени класса, а полное имя класса в данном случае следующее: `JHtmlString` (напомним, что регистр в нем особого значения не имеет). Вторая половина данного аргумента обозначает вызываемый метод (в данном случае — `truncate()`). А остальные аргументы метода с синтаксисом подчеркивания передаются непосредственно вызываемому методу. В рассматриваемом здесь примере в качестве этих аргументов передаются вводный текст и его максимальная длина.

Таким образом, метод `JHtml::_()` *автоматически* загружает нужный класс! Это означает, что нам больше не нужно беспокоиться о загрузке класса, а следовательно, мы можем исключить строку кода с вызовом метода `JLoader::register()` из файла нашей компоновки. В листинге 4.10 приведен окончательный вариант кода из первого примера замещения компоновки с использованием синтаксиса подчеркивания, выделенного в соответствующей строке кода полужирным.

Листинг 4.10. Окончательный вариант компоновки с использованием синтаксиса подчеркивания

```
<?php
/**
 * @version       $Id: default.php 20196 2011-01-09 02:40:25Z ian $
 * @package       Joomla.Site
 * @subpackage    mod_articles_latest
 * @copyright     Copyright (C) 2005 - 2011 Open Source Matters, Inc. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

// непосредственный доступ отсутствует
defined('_JEXEC') or die;
?>
<h1>My First Override</h1>
<ul class="latestnews<?php echo $moduleclass_sfx; ?>">
<?php foreach ($list as $item) : ?>
  <li>
```

```
    <a href="<?php echo $item->link; ?>">
      <?php echo $item->title; ?></a>
      <p>
        <?php echo JHtml::_('string.truncate', strip_tags($item->
introtext), 53);?>
      </p>
  </li>
<?php endforeach; ?>
</ul>
```

Как видите, в этом коде отсутствует строка с вызовом метода `JLoader::register()`. Но дает он такой же результат, как и прежде.

В этом разделе мы рассмотрели немало важных тем, но весь его материал сводится к следующему.

1. Мы создали новую подпапку для хранения файла замещения компоновки в нашем новом шаблоне и скопировали в нее файл стандартной компоновки.
2. Мы видоизменили файл замещения компоновки в соответствии с нашими потребностями. В данном случае мы просто добавили немного предварительно просматриваемого текста под заголовком каждой статьи на странице.

Где бы теперь ни отображался данный модуль с помощью шаблона `beez_20_copy`, для этой цели будет использоваться файл видоизмененной компоновки вместо стандартной.

Изменение внешнего вида компонента регистрации пользователя

Компоновки компонентов можно изменять таким же образом, как и компоновки модулей. Допустим, нам требуется изменить компоновку компонента для регистрации пользователя в пользовательской части веб-сайта, чтобы включить в нее соглашение, которое пользователь должен принять перед тем, как зарегистрироваться на веб-сайте.

Наша первая задача — найти файл компоновки данного компонента и скопировать его в соответствующую подпапку папки шаблона `beez_20_copy`. Регистрация пользователя пользовательской части веб-сайта происходит в компоненте `com_users`, а его компоновка находится в папке `components/com_users/views/registration/tmpl`, где хранятся четыре файла, как показано на рис. 4.15. Так, в файле `default.php` содержится форма регистрации пользователя, а в файле `complete.php` — компоновка экрана, который появляется по завершении регистрации.

Структура папок для хранения замещений компоновок компонентов аналогична структуре папок, создаваемой для хранения замещений компоновок модулей. Следовательно, нам требуется создать подпапку `templates/beez_20_copy/html/<имя_компонента>/<имя_представления>`. В рассматриваемом здесь примере это будет подпапка `beez_20_copy/html/com_users/registration`. Далее мы должны скопировать файл `default.php` из исходной папки `components` в новую папку. В итоге мы получим структуру папок, аналогичную приведенной на рис. 4.16.

Как пояснялось ранее в отношении модулей, файлы замещения их компоновки располагаются в подпапке с именем конкретного модуля. А с компонентами дело обстоит

немного сложнее. У них, как правило, имеется несколько представлений с собственными исходными компоновками. Поэтому нам придется сначала создать подпапку для каждого представления в папке компонента, а затем разместить в ней файлы замещений компоновок. А кроме того, нужно не забыть скопировать пустые файлы `index.html` в папки с замещениями компоновок, чтобы не дать посторонним лицам просматривать их.



Рис. 4.15. Файлы компоновки регистрации пользователя

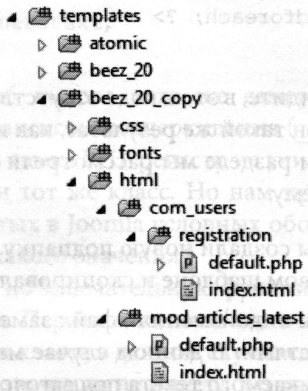


Рис. 4.16. Структура папок для хранения файлов компоновки регистрации пользователя

Далее нам нужно убедиться в том, что выполняется файл замещенной, а не стандартной компоновки. Для этого мы можем снова добавить некоторый текст в файл замещения компоновки и проверить, что он отображается именно так, как и предполагалось. С этой целью в код, приведенный в листинге 4.11, добавлен элемент разметки `h1` с заголовком `My User Registration Override` (Мое замещение компоновки регистрации пользователя); соответствующая строка кода выделена полужирным.

Листинг 4.11. Новый заголовок для проверки выполнения файла замещения компоновки

```
<div class="registration<?php echo $this- >pageclass_sfx?>">
<h1>My User Registration Override</h1>
<?php if ($this->params->get('show_page_heading')) : ?>
  <h1><?php echo $this->escape($this->params->get('page_heading')); ?></h1>
<?php endif; ?>
```

Если выбрать из меню команду Home⇨Create an Account (Домой⇨Создать учетную запись), при условии, что в Joomla установлены выборочные данные, то загрузится форма регистрации пользователя. И если наш шаблон `beez_20_copy` определен как выбираемый по умолчанию, то видоизмененный нами файл компоновки отобразит результат, приведенный на рис. 4.17.

А теперь, когда мы убедились, что все установлено правильно, можно приступать к специальной настройке файла замещения компоновки. Как упоминалось выше, в данном примере нам требуется, чтобы пользователь принял предлагаемое ему соглашение, прежде чем зарегистрироваться. Если проанализировать код в файле компоновки, то в самом его конце можно заметить определение кнопки Register (Зарегистрироваться), выделенное полужирным в листинге 4.12.

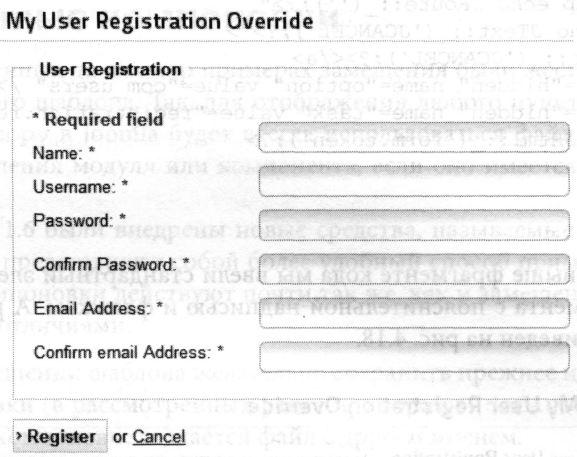


Рис. 4.17. Результат замещения компоновки регистрации пользователя с новым заголовком

Листинг 4.12. Конец файла компоновки

```
<?php endforeach;?>
  <div>
    <button type="submit" class="validate" ><?php echo
JText::_('JREGISTER');?></button>
    <?php echo JText::_('COM_USERS_OR');?>
    <a href="<?php echo JRoute::_('');?>"
title="<?php echo JText::_('JCANCEL');?>"><?php echo JText::_('JCANCEL');?></a>
      <input type="hidden" name="option" value="com_users" />
      <input type="hidden" name="task" value="registration.register" />
      <?php echo JHtml::_('form.token');?>
   </div>
</form>
</div>
```

Нам нужно ввести в компоновку регистрационной формы флажок принятия соглашения перед кодом определения кнопки Register, выделенным полужирным в листинге 4.12. Необходимые для этого изменения в файле компоновки выделены полужирным в листинге 4.13.

Листинг 4.13. Компоновка регистрационной формы с флажком принятия условий обслуживания

```
<?php endforeach;?>
  <div>
    <fieldset>
      <legend>Terms of Service</legend>
        <p><input type="checkbox" /> I agree to the terms of service for
this website.</p>
    </fieldset>
    <button type="submit" class="validate" ><?php echo
JText::_('JREGISTER');?></button>
    <?php echo JText::_('COM_USERS_OR');?>
```

```
    <a href="<?php echo JRoute::_('');?>"
title="<?php echo JText::_('JCANCEL');?>">
<?php echo JText::_('JCANCEL');?></a>
      <input type="hidden" name="option" value="com_users" />
      <input type="hidden" name="task" value="registration.register" />
      <?php echo JHtml::_('form.token');?>
  </div>
</form>
</div>
```

В приведенном выше фрагменте кода мы ввели стандартный элемент `fieldset` разметки HTML-документа с пояснительной надписью и флажком. А результат внесенных нами корректив приведен на рис. 4.18.

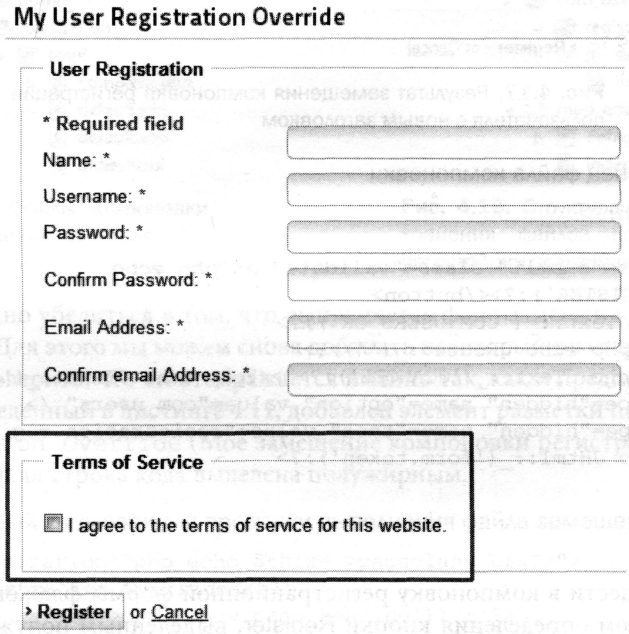


Рис. 4.18. Вид регистрационной формы с условиями обслуживания

Но у обновленной регистрационной формы имеются следующие ограничения.

- От пользователя не требуется устанавливать флажок принятия условий обслуживания перед регистрацией.
- Пояснительный текст к флажку жестко закодирован на английском языке и не переводится при локализации веб-сайта на другие языки мира.

В главе 5 будет представлен подключаемый модуль, проверяющий, установлен ли флажок принятия условий обслуживания. А далее в этой главе мы добавим замещение файла языковой поддержки и видоизменим регистрационную форму таким образом, чтобы весь ее текст можно было перевести на другие языки мира.

Альтернативные компоновки

В двух рассмотренных до сих пор примерах замещения были жестко запрограммированы для конкретного шаблона. Так, для отображения любого пункта меню с помощью шаблона `beez_20_copy` в Joomla будет всегда использоваться файл замещения компоновки для представления модуля или компонента, если оно имеется в папке `beez_20_copy/html`.

В версии Joomla 1.6 были внедрены новые средства, называемые *альтернативными компоновками*. Они представляют собой более удобный способ применения замещений. Альтернативные компоновки действуют почти так же, как и замещения шаблонов, но со следующими двумя отличиями.

- Если при замещении шаблона желательно сохранить прежнее имя файла стандартной компоновки (в рассмотренных ранее примерах — `default.php`), то при альтернативной компоновке создается файл с другим именем.
- Если при замещении шаблона всегда применяется файл замещения, то при альтернативной компоновке выбирается файл, который будет использоваться при создании модуля или пункта меню.

Продемонстрируем особенности альтернативных компоновок, переименовав компоновку модуля последних новостей (Latest News) с `default.php` на `preview.php`. (Напомним, что этот файл находится в папке `templates/beez_20_copy/html/mod_articles_latest`.) Если перейти теперь к компоненту Module Manager и открыть этот модуль для редактирования, то альтернативная компоновка появится в качестве варианта выбора из списка Alternative Layout (Альтернативная компоновка) в области Advanced Options (Дополнительные параметры), как показано на рис. 4.19.

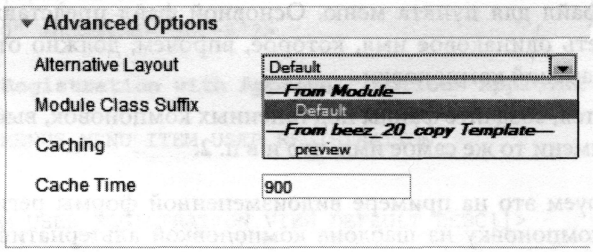


Рис. 4.19. Пример выбора альтернативной компоновки модуля

Аналогичным способом можно создать альтернативные компоновки для статей, контактной информации, лент новостей, веб-ссылок, списков и блогов по категориям. Они действуют таким же образом, как и замещения шаблонов. Если переименовать файл альтернативной компоновки, присвоив ему другое имя, отличающееся от имени стандартной компоновки, он будет воспринят как альтернативная компоновка. Каждый компонент или категорию можно настроить на применение альтернативной компоновки, перейдя к экрану настроек компонента, например, по команде Article Manager⇨Settings (Диспетчер статей⇨Настройки), или же сделать это на уровне отдельных элементов (статей, контактной информации, лент новостей, веб-ссылок или категорий). Дополнительные сведения об альтернативных компоновках можно найти среди документации на Joomla по

адресу http://docs.joomla.org/Layout_Overrides_in_Joomla_1.6#Component_Alternative_Layouts.

Альтернативные компоновки обладают двумя основными преимуществами. Во-первых, для одного и того же модуля можно выбрать несколько альтернативных компоновок, не прибегая ко многим шаблонам. И во-вторых, компоновку можно выбрать даже в том случае, если для отображения страницы вообще не используется шаблон.

К именованию файлов альтернативных компоновок предъявляются следующие минимальные требования.

- Имя файла альтернативной компоновки должно отличаться от имени файла стандартной компоновки.
- В имени файла альтернативной компоновки не должно быть знака подчеркивания (_), поскольку имена файлов со знаком подчеркивания допускаются только для подчиненных компоновок.

Добавление новой компоновки пункта меню

В версии Joomla 1.6 появилась также возможность создавать пункты меню, включающие в себя файл компоновки и XML-файл. Используя XML-файл, можно переопределить параметры настройки пункта меню. Компоновка альтернативного пункта меню создается таким же образом, как и замещение шаблона. Ниже приведена соответствующая процедура.

1. Создайте в папке шаблона подпапку templates/<имя_вашего_шаблона>/html/<имя_компонента>/<имя_представления>/ (т.е. так же, как и для замещения шаблона).
2. Создайте один или несколько файлов исходного кода PHP для представления и один XML-файл для пункта меню. Основной файл представления и XML-файл должны иметь одинаковое имя, которое, впрочем, должно отличаться от имени файла стандартной компоновки.
3. Если требуется, создайте файлы подчиненных компоновок, выбрав для первой половины их имени то же самое имя, что и в п. 2.

Продемонстрируем это на примере видоизмененной формы регистрации пользователя, заменив ее компоновку из шаблона компоновкой альтернативного пункта меню. Для этого выполните следующие действия.

1. Переименуйте файл default.php (в папке templates/beez_20_copy/html/com_users/registration) на approval.php.
2. Скопируйте файл components/com_users/views/registration/tmpl/default.xml в файл templates/beez_20_copy/html/com_users/registration/approval.xml.
3. По желанию добавьте или удалите любые параметры в XML-файле. В данном примере добавляется дополнительный флажок, чтобы с его помощью пользователь мог сообщить, что ему не меньше 18 лет. Для отображения флажка при установке данного параметра в файл компоновки будут внесены соответствующие коррективы.

После копирования упомянутых выше файлов структура папки шаблона beez_20_copy должна принять такой же вид, как и на рис. 4.20.

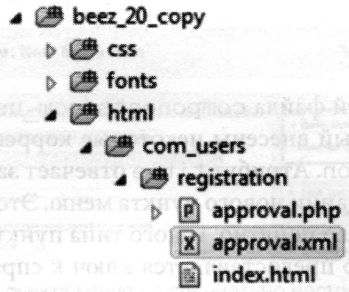


Рис. 4.20. Файлы для создания и компоновки альтернативного пункта меню

Переопределение параметров

Для каждой новой компоновки пункта меню мы создаем новый XML-файл с устанавливаемыми параметрами. Это дает нам возможность специально подбирать параметры для компоновки нового пункта меню. В частности, мы можем свободно добавлять или удалять параметры, а также получать доступ к любым параметрам, добавленным в файл компоновки. Итак, добавим в файл компоновки параметр для отображения или сокрытия второго флажка. Соответствующий код с выделенными полужирным изменениями в XML-файле приведен в листинге 4.14.

Листинг 4.14. Код из XML-файла для ввода альтернативного пункта меню в регистрационную форму пользователя

```
<?xml version="1.0" encoding="utf-8"?>
<metadata>
  <layout title="Registration with Approval" option="Approval">
    <help
      key="JHELP_MENUS_MENU_ITEM_USER_REGISTRATION"
    />
    <message>
      <![CDATA[COM_USER_REGISTRATION_VIEW_DEFAULT_DESC]]>
    </message>
  </layout>

  <!-- Добавить поля в объект параметров для компоновки. -->
  <fields name="params">

    <!-- Основные параметры. -->
    <fieldset name="basic" label="COM_MENUS_BASIC_FIELDSET_LABEL">
      <field
        name="show_age_checkbox"
        type="radio"
        label="Show Age Checkbox"
        description="Show or hide the age checkbox."
        default="0" >
```

```
        <option value="0">Hide</option>
        <option value="1">Show</option>
      </field>
    </fieldset>
  </fields>
</metadata>
```

Этот файл является копией файла `components/com_users/views/registration/tmpl/default.xml`, в который внесены некоторые коррективы. В частности, мы изменили атрибуты `title` и `option`. Атрибут `title` отвечает за отображение заголовка регистрационной формы при создании нового пункта меню. Этот заголовок должен отличаться от стандартного, чтобы было понятно, какого типа пункт меню создается.

В элементе разметки `help` предоставляется ключ к справочной системе, чтобы было понятно, какую именно статью следует загружать при выборе пиктограммы справки. Но поскольку у нас нет отдельного справочного файла для замещения, мы оставим данный элемент без изменения. И последним изменением является добавление элемента `field` для разметки нового параметра кнопки-переключателя, называемого `show_age_checkbox`. У этого параметра имеются два варианта выбора: Hide (Скрывать) и Show (Показывать), причем вариант `Hide` выбирается по умолчанию. При создании нового пункта меню рассматриваемый здесь XML-файл обрабатывается в классах типа `JForm` из библиотеки Joomla для воспроизведения вариантов выбора данного пункта меню.

А теперь посмотрим, как все это будет выглядеть при использовании нового альтернативного типа пункта меню. Если выбрать сначала команду меню Menus⇨Main Menu (Меню⇨Главное меню), а затем пиктограмму New и щелкнуть на кнопке Select, чтобы выбрать поле Menu Item Type (Тип пункта меню), появится список, аналогичный приведенному на рис. 4.21.

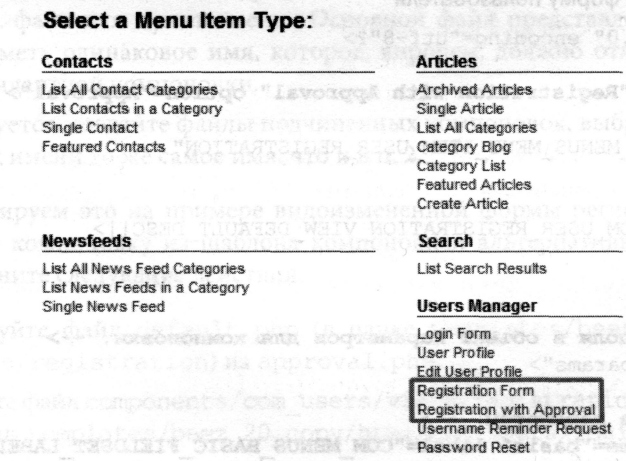


Рис. 4.21. Список типов пунктов меню с новым альтернативным типом пункта меню

Обратите внимание на то, что в этом списке перечислен созданный нами новый альтернативный тип пункта меню `Registration with Approval` (Регистрация с подтверждением). Если выбрать этот тип пункта меню, появится экран New Menu Item (Новый пункт

меню) с кнопкой-переключателем Show Age Checkbox (Показывать флажок возрастных ограничений), как показано на рис. 4.22. Обратите внимание на то, что новый параметр появляется в области Basic Options (Основные параметры).

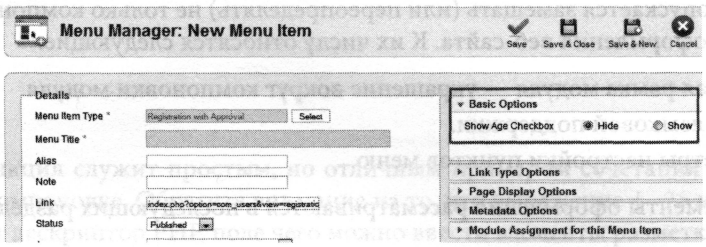


Рис. 4.22. Экран ввода пункта меню со специально настраиваемыми параметрами

А теперь воспользуемся новым параметром для добавления второго флажка в компоновку регистрационной формы по заданному условию. Изменения, внесенные с этой целью, выделены в приведенном ниже фрагменте кода полужирным.

```
<fieldset>
  <legend>Terms of Service</legend>
    <p><input type="checkbox" /> I agree to the terms of service for this
↳website. </p>
    <?php if ($this->params->get('show_age_checkbox')) : ?>
      <p><input type="checkbox" /> I am at least 18 years old. </p>
    <?php endif; ?>
</fieldset>
```

В данном случае мы добавили код для проверки нового параметра `show_age_checkbox`. Он становится доступным автоматически в поле `this->params` посредством метода `get()`. Если установлено значение **0** этого параметра, флажок для указания возраста в регистрационной форме скрывается, а если установлено значение **1** — отображается. Нулевое значение в коде PHP вычисляется как ложное логическое значение (`false`) выражения в условном операторе `if`. В итоге флажок для указания возраста отображается лишь в том случае, если кнопка-переключатель установлена в положение Show. Используя вновь созданный нами альтернативный тип пункта меню, попробуйте сначала создать новый пункт меню `My User Registration` (Моя регистрация пользователя), а затем отобразить скомпонованную регистрационную форму при двух разных значениях параметра настройки этого пункта меню, чтобы убедиться в правильности его функционирования.

О принципе действия замещений

Код, положенный в основу замещений компоновок модулей и компонентов, интересен и поучителен. Но анализировать его целесообразно лишь после того, как станет понятнее нормальное функционирование самих модулей и компонентов. Поэтому отложим рассмотрение кода, реализующего компоновки модулей, до главы 6, а кода, реализующего альтернативные пункты меню, — до главы 8.

Замещение других элементов оформления веб-сайта

В Joomla допускается замещать (или переопределять) не только компоновки, но и другие элементы оформления веб-сайта. К их числу относятся следующие.

- Стилевая рамка модуля — украшение вокруг компоновки модуля.
- Файлы языковой поддержки.
- Параметры настройки пунктов меню.

Все эти элементы оформления рассматриваются в последующих разделах.

Стилевая рамка модуля: добавление нового стиля оформления модуля

Стили оформления вообще и стилевая рамка модуля в частности рассматривались ранее в этой главе. А здесь покажем, как они действуют, на примере создания нового стиля оформления модуля для специально настраиваемого шаблона. Для этого придется выполнить следующие действия.

1. Создать новую функцию `modChrome()` в файле `modules.php` для специально настраиваемого шаблона.
2. Воспользоваться новым стилем при объявлении места расположения модуля в шаблоне с помощью элемента разметки `jdoc:include` в файле `index.php`.
3. Назначить модуль для этого места расположения.

Ради экономии времени скопируем уже имеющуюся функцию `modChrome()` из файла `templates/system/html/modules.php`, где она называется `modChrome_table()`, а затем вставим ее в конце файла `templates/beez_20_copy/html/modules.php`. Далее переименуем скопированную функцию на `modChrome_beez20_copyFramedTable()`. В итоге завершающий фрагмент кода из файла `templates/beez_20_copy/html/modules.php` будет выглядеть так, как показано в листинге 4.15.

Листинг 4.15. Новая функция `modChrome_beez20_copyFramedTable()`

```
/*
* Пример специально настраиваемой стилевой рамки модуля
* для шаблона beez_20_copy
*/
function modChrome_beez20_copyFramedTable ($module, &$params, &$attribs)
{ ?>
  <table cellpadding="0" cellspacing="0"
↳class="moduletable<?php echo
↳htmlspecialchars($params->get('moduleclass_sfx')); ?>">
  <?php if ($module- >showtitle != 0) : ?>
    <tr>
      <th>
         <?php echo $module->title; ?>
      </th>
    </tr>
```

```
  <?php endif; ?>
    <tr>
      <td>
         <?php echo $module->content; ?>
      </td>
    </tr>
  </table>
  <?php
}
```

Данная функция служит простым, но отличным примером сочетания кода HTML и PHP в одной компоновке. Обратите внимание на то, что выражение { ?> в пятой строке кода закрывает дескриптор PHP, поле чего можно ввести элементы разметки HTML. Сначала в этой функции создается элемент `table` разметки HTML-документа, а затем проверяется, требуется ли отобразить заголовок модуля. Если требуется, то создается элемент `th` разметки табличного заголовка. И наконец, содержимое модуля выводится в строке таблицы (элемент разметки `tr`) и ячейке таблицы (элемент разметки `td`).

В отношении данной функции необходимо также отметить следующее. Как упоминалось ранее, это функция, а не метод класса PHP, и поэтому она располагается в самом конце файла и не заключена в фигурные скобки определения класса. Поэтому непременно введите эту функцию в самом конце файла `modules.php`, т.е. после самой последней закрывающей фигурной скобки. Следует также иметь в виду, что наименование шаблона `beez20_copy` включено как составная часть в имя данной функции. Это очень важно сделать на тот случай, если данный шаблон предполагается использовать на веб-сайтах с другими шаблонами.

Как и следовало ожидать, в языке PHP не разрешается именовать одну и ту же функцию дважды, поскольку будет неясно, какой именно версией функции следует воспользоваться. При добавлении нового шаблона в оформление веб-сайта мы заранее не знаем, какие именно функции `modChrome()` могли быть введены в файлы `modules.php` имеющихся шаблонов. Поэтому включив название шаблона в имя функции, мы можем быть вполне уверены в том, что функция с аналогичным именем не встретится в другом шаблоне, а следовательно, не вызовет ошибку при выполнении кода PHP. Такой способ добавления уникального префикса к имени элемента программы иногда еще называется *разделением на пространства имен*. Аналогичным способом мы воспользуемся в дальнейшем при создании ключей для файлов языковой поддержки.

А теперь видоизменим новый стиль таким образом, чтобы заключить таблицу в рамку шириной **5** пикселей. С этой целью добавим атрибут `border` в элемент `table` разметки HTML-документа, как выделено полужирным в приведенном ниже фрагменте кода.

```
function modChrome_beez20_copyFramedTable($module, &$params, &$attribs)
{ ?>
  <table cellpadding="0" cellspacing="0" border="5"
↳class="moduletable<?php echo
↳htmlspecialchars($params->get('moduleclass_sfx')); ?>">
```

Имея в своем распоряжении новый стиль, мы должны теперь внести коррективы в файл `index.php` нашего шаблона, чтобы выделить место для применения данного стиля. С этой целью отредактируем файл `templates/beez_20_copy/index.php`, добавив в нем новое место расположения `position-7a`. Напомним, что места расположения

модулей добавляются с помощью элемента разметки (или дескриптора) `jdoc:include`. Ниже приведен фрагмент кода с изменениями, внесенными в файл `index.php` и выделенными полужирным.

```
<div class="left <?php if ($showRightColumn==NULL){ echo 'leftbigger';}
↳?>" id="nav" >

  <jdoc:include type="modules" name="position- 7" style="beezDivision"
↳headerLevel="3" />
  <jdoc:include type="modules" name="position-7a"
↳style="beez20_copyFramedTable" />
  <jdoc:include type="modules" name="position-4" style="beezHide"
↳headerLevel="3" state="0 " />
```

Если бы это было настоящее место расположения модуля в шаблоне, нам бы потребовалось добавить его и в файле `templateDetails.xml`. Но для целей рассматриваемого здесь примера этого не требуется.

И наконец, нам нужно внести коррективы в сам модуль, чтобы использовать его на месте `position-7a`, выделенном в шаблоне. С этой целью перейдем к компоненту Module Manager (Диспетчер модулей) и изменим значение места расположения, как показано на рис. 4.23. Но поскольку мы не добавили место расположения в файл `templateDetails.xml`, то оно и не появится в списке, а следовательно, нам придется ввести его вручную. Кроме того, изменим назначение меню модуля (`Menu Assignment`), чтобы отображать его на всех страницах (`On All Pages`).



Рис. 4.23. Указание нового места расположения модуля в диспетчере модулей

Если теперь отобразить начальную страницу веб-сайта, то модуль последних новостей появится в новой стилевой рамке, как показано на рис. 4.24. (Следует иметь в виду, что и в данном примере используется стандартная, а не замещенная компоновка, поскольку файл `default.php` был переименован в `preview.php` в папке `templates/beez_20_copy/html/mod_articles_latest`.)

Далее воспользуемся атрибутами стилевой рамки модуля, чтобы предоставить веб-разработчикам возможность установить ширину обрамления в дескрипторе `jdoc:include`. С этой целью внесем в код функции из листинга 4.15 изменения, выделенные ниже полужирным.

```
{ ?>
  <table cellpadding="0" cellspacing="0"
↳border="<?php echo $attribs['border'] ?>"
↳class="moduletable<?php
↳echo htmlspecialchars($params->get('moduleclass_sfx')); ?>">
```

В этом фрагменте кода значение атрибута `border` устанавливается с помощью переменной `$attribs['border']` в коде PHP. Это значение должно быть затем передано рассматриваемой здесь функции с учетом атрибутов, указанных в элементе разметки `jdoc:include`. Для этого придется внести в файл `index.php` шаблона изменения, выделенные ниже полужирным:

```
<jdoc:include type="modules" name="position-7a"
➥style="beez20_copyFramedTable"
➥border="2" headerLevel="3" />
```

В данном случае установлено значение **2** пикселя для атрибута `border`. А для того чтобы варьировать ширину обрамления в любом месте шаблона, где применяется стиль `beez20_copyFramedTable`, достаточно установить атрибут `border` в соответствующем элементе разметки `jdoc:include`.

Но что, если оставить атрибут `border` за пределами элемента разметки `jdoc:include`? Прежде чем решиться на подобный эксперимент, следует выбрать сначала команду **Global Configuration⇨Server** (Глобальная настройка⇨Сервер) из меню в административной части веб-сайта, а затем вариант **Maximum** из раскрывающегося списка **Error Reporting** (Сообщение об ошибках), как показано на рис. 4.25.

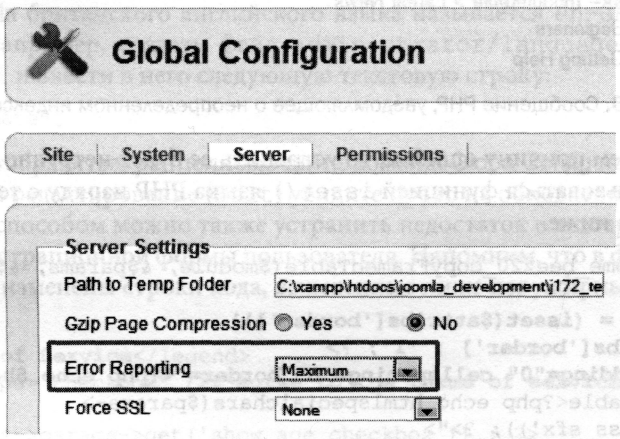


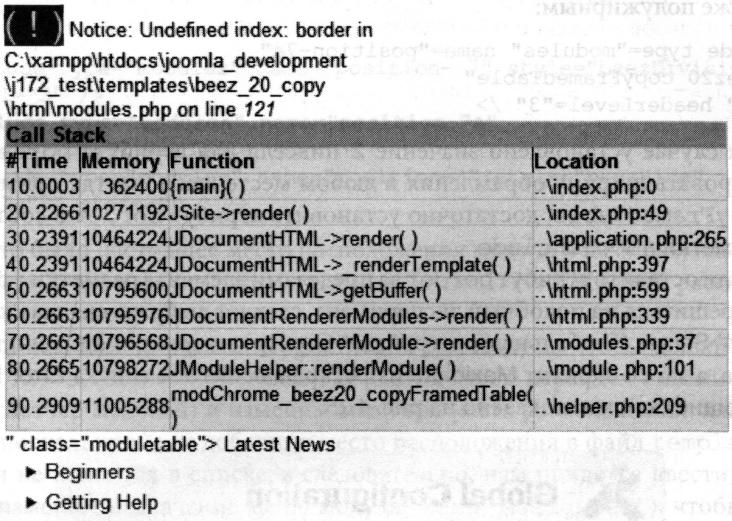
Рис. 4.25. Установка режима сообщения об ошибках на сервере

Настоятельно рекомендуется устанавливать упомянутый выше режим сообщения об ошибках, приступая всякий раз к написанию кода для Joomla, и ниже будет показано, для чего это вообще нужно. Для этого придется вернуться к файлу `index.php` шаблона и удалить атрибут `border` из элемента `jdoc:include`, как показано ниже.

```
<jdoc:include type="modules" name="position-7a"
➥style="beez20_copyFramedTable" />
```

Если теперь мы отобразим рассматриваемый здесь модуль, то получим уведомляющее сообщение языка PHP, как показано на рис. 4.26. Дело в том, что мы ссылаемся на элемент массива, `$attribs['border']`, который не определен. Если бы мы установили обычный режим сообщения об ошибках, то не получили бы подобное предупреждение.

Поэтому при написании кода очень важно добиваться максимальной его надежности и безошибочности. И для этого, в частности, следует сначала объявлять переменные, а затем попытаться воспользоваться ими.


Notice: Undefined index: border in C:\xampp\htdocs\joomla_development\j172_test\templates\beez_20_copy\html\modules.php on line *121*

Call Stack

#Time	Memory	Function	Location
1 0.0003	362400	{main}( )	..\index.php:0
2 0.2266	10271192	JSite->render( )	..\index.php:49
3 0.2391	10464224	JDocumentHTML->render( )	..\application.php:265
4 0.2391	10464224	JDocumentHTML->_renderTemplate( )	..\html.php:397
5 0.2663	10795600	JDocumentHTML->getBuffer( )	..\html.php:599
6 0.2663	10795976	JDocumentRendererModules->render( )	..\html.php:339
7 0.2663	10796568	JDocumentRendererModule->render( )	..\modules.php:37
8 0.2665	10798272	JModuleHelper::renderModule( )	..\module.php:101
9 0.2909	11005288	modChrome_beez20_copyFramedTable( )	..\helper.php:209

" class="moduletable"> Latest News

- Beginners
- Getting Help

Рис. 4.26. Сообщение PHP, уведомляющее о неопределенном индексе массива

Если мы поймем причину ошибки, то устранить ее будет нетрудно. Для этого можно, например, воспользоваться функцией `isset()` языка PHP наряду с тернарным оператором, как показано ниже.

```
function modChrome_beez20_copyFramedTable($module, &$params, &$attribs)
{ ?>
  <?php $border = (isset($attribs['border']))
? (int) $attribs['border'] : '1'; ?>
  <table cellpadding="0" cellspacing="0" border="<?php echo $border ?>"
class="moduletable<?php echo htmlspecialchars($params->
get('moduleclass_sfx')); ?>">
```

Тернарный оператор языка PHP возвращает значение, исходя из истинности или ложности условия. Ниже приведен его синтаксис.

```
<условие> ? <значение, если условие истинно> : < значение, если условие ложно>
```

В данном примере вычисляется выражение `isset($attribs['border'])`. Если оно истинно, то в переменной `$border` устанавливается целочисленное значение элемента массива `$attribs['border']`, а иначе — значение **1**. Оператор `(int)` используется для того, чтобы значение данного атрибута всегда было целочисленным. Если же пользователь по ошибке установит нецелочисленное значение атрибута `border`, оно все равно будет приведено к целочисленному. Следует заметить, что аналогичного результата можно было бы добиться и с помощью условных операторов `if/then`, но тернарный оператор более компактен и позволяет выполнить все необходимые действия в одной строке кода.

Языковые замещения: добавление перевода в замещение

В версии Joomla 1.6 внедрено новое удобное средство для специальной настройки текста, отображаемого в пользовательском интерфейсе веб-сайта. В частности, для размещения файлов языковой поддержки предоставляются две новые папки: `language/overrides` в пользовательской части и `administrator/language/overrides` в административной части.

Обратимся к конкретному примеру. Если заглянуть в файл языковой поддержки `administrator/language/en-GB/en-GB.ini`, то в нем можно обнаружить следующую текстовую строку:

```
JACTION_EDIT="Edit"
```

Языковой ключ `JACTION_EDIT` используется в правах доступа системы ACL, например, по команде Global Configuration⇨Permissions (Глобальная настройка⇨Полномочия), для отображения права на редактирование (`Edit`).

Мы можем переопределить любой языковой ключ в административной части, создав файл замещения под соответствующим именем и введя в него какие угодно языковые ключи. Для именования такого файла приняты следующие условные обозначения: `xx-XX.override.ini`, где `xx-XX` — требуемое обозначение языка. Так, файл поддержки исходно принятого в Joomla британского английского языка называется `en-GB.override.ini`.

Попробуйте, например, создать файл `administrator/language/overrides/en-GB.override.ini` и ввести в него следующую текстовую строку:

```
JACTION_EDIT="Update"
```

Если затем вы отобразите экран полномочий по команде Global Configuration⇨Permissions, то вместо права на редактирование (`Edit`) увидите право на обновление (`Update`).

Аналогичным способом можно также устранить недостаток в альтернативном пункте меню выбора регистрационной формы пользователя. Напомним, что в файле компоновки этой формы были изменены строки кода, выделенные ниже полужирным.

```
<fieldset>
  <legend>Terms of Service</legend>
    <p><input type="checkbox" />I agree to the terms of service for this
↳website. </p>
  <?php if ($this->params->get('show_age_checkbox')) : ?>
    <p><input type="checkbox" />I am at least 18 years old. </p>
  <?php endif; ?>
</fieldset>
```

В трех местах этого фрагмента кода текст отображается только на английском языке. Но в практике разработки программ для Joomla такое жесткое программирование отображаемого текста не приветствуется, поскольку у Joomla немало иноязычных пользователей. В Joomla имеется удобное средство, называемое языком отладки (Debug Language) и помогающее проверять наличие перевода всех текстовых строк на избранный язык мира. Если выбрать команду меню Global Configuration⇨System (Глобальная настройка⇨Система), то в области Debug Settings (Параметры отладки) появившегося экрана можно обнаружить параметры выбора режимов Debug System (Система отладки) и Debug Language (Язык отладки), как показано на рис. 4.27.

Режим Debug System полезен для выявления неполадок в базе данных посредством запросов к ней, что и будет продемонстрировано в главе 11. А в режиме Debug Language мы

можем видеть, переведены ли текстовые строки. Если включить режим Debug Language, установив кнопку-переключатель в положение Yes, а затем отобразить альтернативный пункт меню выбора регистрационной формы пользователя, то она будет выглядеть так, как показано на рис. 4.28.

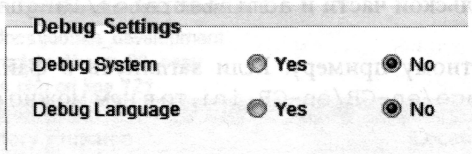


Рис. 4.27. Параметры отладки на экране глобальной настройки

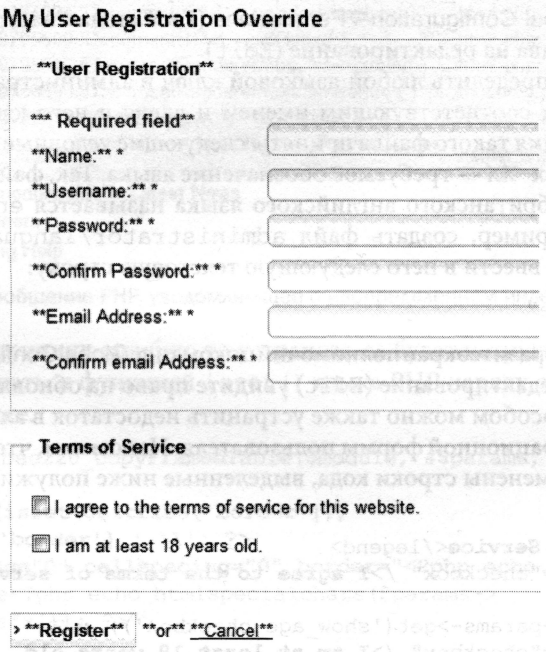


Рис. 4.28. Замещенная компоновка регистрационной формы пользователя в активизированном режиме языка отладки

Все переведенные текстовые строки предваряются и завершаются двумя знаками звездочки (**). Обратите внимание на то, что три ранее введенные строки (`"Terms of Service"`, `"I Agree . . . "` и `"I Am . . . "`) не обозначены знаками **. Это означает, что они не могут быть переведены. Но используя файлы языковых замещений, можно без особого труда локализовать замещение компоновки на многие языки мира. Для этого достаточно заменить конкретную текстовую строку вызовом метода `JText::_('КЛЮЧ_ТРЕБУЮЩЕГО_ТЕКСТА')` в коде, как выделено полужирным ниже.

```
<fieldset>
  <legend><?php echo JText::_('BEEZ_20_COPY_TERMS_OF_SERVICE')?></legend>
    <p><input type="checkbox" /> <?php echo JText::_('BEEZ_20_COPY_AGREE')?>
```

```
</p>
    <?php if ($this->params->get('show_age_checkbox')) : ?>
      <p><input type="checkbox" /> <?php echo JText::_('BEEZ_20_COPY_AGE')?>
</p>
  <?php endif; ?>
</fieldset>
```

В качестве аргумента метода `JText::_()` указывается так называемый языковой ключ. Значение этого ключа обнаруживается средствами Joomla в одном из загруженных файлов языковой поддержки с расширением `.ini` и затем возвращается из данного метода.

Для языковых ключей установлены следующие правила. Во-первых, они не могут содержать пробелы и ряд зарезервированных слов (`null`, `yes`, `no`, `true`, `false`, `on`, `off` и `none`), поскольку для обработки файлов языковой поддержки в Joomla применяется функция `parse_ini_file()` языка PHP. И во-вторых, каждый языковой ключ должен быть уникальным, иначе может быть неумышленно изменено не то текстовое значение, которое предполагалось изначально. Обеспечить уникальность языкового ключа проще всего, включив имя расширения в его состав. Но поскольку в рассматриваемом здесь примере мы не пользуемся расширением, то вместо него мы указали в первой части каждого ключа имя нашего шаблона, обеспечив тем самым уникальность ключа в пределах шаблона. Это еще один пример использования простого пространства имен для создания уникального имени.

Значения языковых ключей должны быть заключены в двойные кавычки. Если же требуется заключить в двойные кавычки текстовую строку на избранном языке мира, то вместо двойных кавычек можно ввести `"` или `_QQ_`. Соответствующие примеры такого оформления текстовых сток можно найти в базовых файлах языковой поддержки Joomla.

А теперь рассмотрим видоизмененную нами регистрационную форму пользователя, приведенную на рис. 4.29. Как видите, значения ключей заключены в двойные вопросительные знаки (**??**). Это означает, что методу `JText::_()` не удалось найти переданные ему ключи в файле языковой поддержки. Но именно это нам и требуется, поскольку указывает на однозначность ключей и их отсутствие во всех загруженных файлах языковой поддержки.

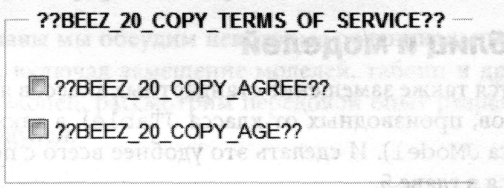


Рис. 4.29. Частичный вид регистрационной формы пользователя с отсутствующими языковыми ключами

Итак, нам осталось лишь создать файл языкового замещения, введя в него новые языковые ключи. А поскольку данная регистрационная форма отображается в пользовательской части, то нам нужно создать файл замещения для британского английского языка (`en-GB`) именно в этой части веб-сайта. Этот файл будет полностью называться `language/overrides/en-GB.override.ini` и содержать следующее:

```
; Пример файла языкового замещения в пользовательской части веб-сайта
; Ключи для файла компоновки templates/beez_20_copy/html/com_users/registration/
approval.php
```

```
BEEZ_20_COPY_TERMS_OF_SERVICE="Terms of Service"
BEEZ_20_COPY_AGE="I am at least 18 years old."
BEEZ_20_COPY_AGREE="I agree to the terms of service for this site."
```

Если теперь отобразить регистрационную форму пользователя, то весь ее текст, в том числе и введенный нами ранее, будет обрамлен двойными вопросительными знаками, указывающими на то, что текст данной формы переведен на избранный язык мира, а по существу, взят из файла языковой поддержки. На рис. 4.30 показана видоизмененная часть регистрационной формы пользователя с текстом на британском английском языке.

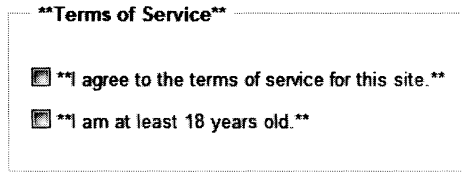


Рис. 4.30. Частичный вид регистрационной формы пользователя с переведенным текстом

Языковыми замещениями можно пользоваться с двумя целями.

- Изменить текст для имеющегося языкового ключа.
- Предоставить переведенный текст для языкового замещения.

Вместе с расширением, как правило, создаются и новые, специально предназначенные для него файлы языковой поддержки, а не используются файлы языковых замещений. Подробнее о файлах языковой поддержки подключаемых модулей, компонентов и модулей речь пойдет в последующих главах.

В версии Joomla 2.5 появилась возможность создавать и редактировать языковые замещения во вкладке Overrides (Замещения) диспетчера языков, доступного по команде меню Extensions⇨Language Manager (Расширения⇨Диспетчер языков) в административной части. Это делается точно так же, как и продемонстрировано выше вручную.

Замещения таблиц и моделей

В Joomla допускается также замещение стандартных классов языка PHP, применяемых для таблиц (подклассов, производных от класса `JTable`), а также моделей (подклассов, производных от класса `JModel`). И сделать это удобнее всего с помощью подключаемого модуля, как поясняется в главе 5.

Резюме

В этой главе представлен краткий обзор шаблонов, применяемых в Joomla, а также продемонстрированы многочисленные способы замещения стандартных компоновок в Joomla. Все это требует минимального программирования, не прибегая к написанию новых расширений.

Подобным образом можно специально настроить немало основных функциональных средств в Joomla. Но для того чтобы полностью раскрыть потенциал расширения функциональных возможностей Joomla, необходимо создавать собственные расширения. Именно об этом и пойдет речь в последующих главах данной книги.

5

Расширение Joomla подключаемыми модулями

В этой главе исследуется принцип действия подключаемых модулей, а также различные виды событий, запускающих подключаемые модули на выполнение. Сначала мы рассмотрим основные подключаемые модули, а затем разработаем свой собственный подключаемый модуль на основе созданной раньше формы регистрации пользователей.

Далее мы создадим упакованный архив, чтобы устанавливать и использовать наш подключаемый модуль как расширение на любом веб-сайте Joomla. После этого введем ряд параметров в подключаемый модуль и рассмотрим класс `JForm`, в котором эти параметры обрабатываются. Кроме того, мы добавим файл языковой поддержки для организации перевода текстовых надписей в нашем подключаемом модуле.

В заключение этой главы мы обсудим некоторые потенциальные возможности подключаемых модулей, включая замещение моделей, таблиц и других стандартных классов Joomla. И наконец, рассмотрим передовой опыт разработки и применения подключаемых модулей.

Что такое подключаемый модуль

Подключаемый модуль — это просто программа на PHP, выполняющаяся в один или несколько предварительно заданных моментов в цикле выполнения команд Joomla. Такие моменты называются событиями и инициируются в Joomla.

Подключаемый модуль может быть очень простым, устанавливая, в частности, значение в поле перед сохранением в базе данных, или же очень сложным, преобразуя, например, в другой формат все URL, присутствующие в документе. Подключаемые модули можно даже использовать для замещения стандартных базовых классов. Они тесно связаны с событиями, и поэтому для уяснения принципа

действия подключаемых модулей очень важно понять назначение предварительно заданных событий в Joomla и как их формировать самостоятельно.

Принцип действия подключаемых модулей

Действие подключаемых модулей можно разделить на три стадии.

1. Один или более файл подключаемых модулей включается в текущий сценарий, что, как правило, делается с помощью метода `JPluginHelper::importPlugin()`. Подключаемые модули зачастую представлены объявлениями классов, и поэтому код на данной стадии не исполняется.
2. Инициируется событие, для чего, как правило, вызывается метод `$dispatcher->trigger()`, где `$dispatcher` — объект типа `JDispatcher`. Каждому событию присваивается отдельное имя, например `onBeforeInitialise` или `onContentBeforeSave`.
3. Код обработки события осуществляет поиск метода, совпадающего с именем события, в любых подключаемых модулях, активизированных и загруженных в настоящий момент. Если такой метод обнаружен, он выполняется.

Загрузка и исполнение сценариев на PHP, в которых объявляются классы

Большинство подключаемых модулей в Joomla состоят только из объявления класса. Это означает, что в них объявляется класс и определяются методы этого класса, а помимо объявления класса вообще отсутствуют какие-либо строки кода.

Когда загружается файл такого типа (например, по команде `require_once` языка PHP), в рабочей области оперативной памяти создается новый класс со всеми его методами. А отдельный метод выполняется только при исполнении строки кода, из которой он вызывается.

Именно поэтому метод `JPluginHelper::importPlugin()` подготавливает подключаемый модуль к выполнению, но, как правило, он не исполняет сам код. Методы подключаемого модуля выполняются только при инициировании события, например, методом `$dispatcher->trigger()`.

Условные обозначения подключаемых модулей

Подключаемые модули, обнаруживаемые методом `JPluginHelper::importPlugin()`, должны соответствовать принятым условным обозначениям имен файлов и классов. Имена папок и файлов для хранения подключаемых модулей, обозначаются следующим образом:

```
plugins/<тип пм>/<имя пм>/<имя пм>.php
```

где пм — подключаемый модуль.

Так, файл подключаемого модуля SEF обозначается как `plugins/system/sef/sef.php`. А XML-файл в подключаемом модуле имеет такое же имя, например, `plugins/system/sef/sef.xml`. Более подробно XML-файл будет рассматриваться далее в главе, когда речь пойдет об упаковке подключаемого модуля в расширение.

Имя класса из подключаемого модуля указывается в соответствии со следующими условными обозначениями:

```
"plg" + <тип пм> + <имя файла пм>
```

где пм — подключаемый модуль.

Например, имя класса из подключаемого модуля SEF обозначается как `plgSystemSEF`.

Типы подключаемых модулей и место для их ввода

Подключаемые модули выполняются при инициировании связанных с ними событий, которые определяются в Joomla иначе, чем в событийно-управляемых программах. В событийно-ориентированном программировании соответствующая программа ожидает наступления определенного события, которое обычно связано с некоторыми действиями пользователя, например, щелчком кнопкой мыши или нажатием клавиш при вводе с клавиатуры. А в Joomla события следует рассматривать в качестве контрольных пунктов на различных путях в цикле выполнения команд. Всякий раз, когда такой контрольный пункт достигается в цикле выполнения команд, инициируется событие. События твердо установлены, хотя различные события инициируются в зависимости от вида цикла исполнения. Рассмотрим вкратце разновидности событий.

Аутентификация

Для аутентификации пользователя имеется только одно событие под названием `onUserAuthenticate`. Это событие инициируется всякий раз, когда пользователь пытается зарегистрироваться в пользовательской или административной части веб-сайта.

Контрольное изображение

Контрольное изображение представляет собой способ предотвратить несанкционированную массовую рассылку навязчивых сообщений (как правило, в рекламных целях). С этой целью от пользователя требуется ввести некоторый образец текста, намеренно представленный искаженными буквами. Такой способ проверки, действительно ли пользователь является человеком, а не автоматом, реализуется с помощью следующих трех событий: `onInit`, `onDisplay` и `onCheckAnswer`.

Содержимое

События, связанные с содержимым, инициируются при отображении или редактировании содержимого. К содержимому относятся статьи, контактная информация и прочие сведения.

Редакторы

В Joomla редакторы реализуются в качестве подключаемых модулей, хотя они и не подпадают под категорию подключаемых модулей, рассматриваемых в этой главе. Для того чтобы добавить новый редактор в Joomla, требуются основательные знания и опыт программирования на JavaScript, да и дело это непростое. И хотя добавление редакторов в этой книге подробно не рассматривается, тем не менее в ней показывается, как пользоваться редакторами при заполнении полей форм.

Editors-XTD

Editors-XTD — это подключаемые модули, используемые для создания кнопок или ссылок, появляющихся ниже редакторов, например, `Image` (Изображение), `Pagebreak` (Перейти на новую страницу) или `Read More` (Читать дальше). Для этих подключаемых модулей предусмотрено только одно событие под названием `onDisplay`.

Расширение

Этот тип подключаемого модуля внедрен в версии Joomla 1.6. События, связанные с расширениями, инициируются при установке, удалении, редактировании или сохранении расширений в компонентах Module, Plugin, Template или Language Manager.

Поиск

В подключаемых модулях поиска реализуются поисковые функции Joomla. Основные подключаемые модули данного типа разделяются на категории, контактную информация, содержимое, ленты новостей и веб-ссылки. С ними связаны следующие события: `onContentSearchAreas` и `onContentSearch`. В частности, событие `onContentSearchAreas` служит для создания массива элементов искомого содержимого, а событие `onContentSearch` — для поиска содержимого по отдельным его видам. Разработчики расширений могут подключать к ним модули поиска, чтобы организовать поиск в разрабатываемых компонентах средствами Joomla.

Интеллектуальный поиск

Подключаемые модули интеллектуального поиска находятся в папке `plugins/finder`. Эти подключаемые модули служат для индексирования содержимого веб-сайта с целью реализовать функцию интеллектуального поиска (Smart Search). Для каждого вида содержимого (контактной информации, лент новостей, обычного содержимого и веб-ссылок) предоставляется отдельный подключаемый модуль, который может быть активизирован для индексирования данного вида содержимого. С этими подключаемыми модулями связаны следующие события: `onFinderAfterDelete`, `onFinderAfterSave`, `onFinderBeforeSave`, `onFinderCategoryChangeState` и `onFinderChangeState`.

Система

Системные подключаемые модули предоставляют события, инициируемые в каждом цикле выполнения команд Joomla. К их числу относится `onAfterInitialise` — первое событие, инициируемое в Joomla. Все системные события связаны с методами `render()`, `dispatch()` и `route()`. Системные события следует инициировать для тех подключаемых модулей, которые должны запускаться на выполнение в каждом цикле выполнения команд независимо от решаемой задачи.

Пользователь

Пользовательские события инициируются при выполнении двух разных задач. Одна группа событий связана с редактированием пользовательской информации в компоненте User Manager (Диспетчер пользователей). К этой группе относятся события

onUserAfterDelete, onUserAfterSave, onUserBeforeDelete и onUserBeforeSave. А вторая группа событий связана со входом и выходом из веб-сайта. К этой группе относятся события onUserLogin и onUserLogout.

Краткий обзор избранных базовых подключаемых модулей

В подключаемых модулях смущает лишь то, что они слишком сильно разнятся, а общим у них является только порядок их именования. Принимая во внимание это обстоятельство, начнем краткий обзор с нескольких базовых подключаемых модулей. На их примерах можно получить ясное представление о том разнообразии задач, которые можно решать с помощью подключаемых модулей.

Системный подключаемый модуль SEF

В качестве первого примера рассмотрим системный подключаемый модуль SEF. Он представляет собой класс plgSystemSef, объявляемый в файле plugins/system/sef/sef.php и активизирующий функции Joomla с целью использовать удобные для поиска URL (иначе называемые SEF URL).

Где запускается

Рассмотрим сначала, каким образом этот подключаемый модуль запускается на выполнение, а именно: код из класса этого модуля для инициирования связанного с ним события. Подключаемый модуль SEF относится к категории системных и запускается на выполнение при наступлении события onAfterRender. Но прежде чем инициировать это событие, необходимо включить файл данного подключаемого модуля.

Если загружается страница в пользовательской части веб-сайта, то вызывается метод render() из класса JSite, находящегося в файле includes/application.php. Ближе к концу этого метода можно обнаружить следующую строку кода:

```
JPluginHelper::importPlugin('system');
```

В этой строке кода содержится команда для загрузки всех активизированных подключаемых модулей в рабочую область оперативной памяти. (Если подключаемый модуль заблокирован в компоненте Plugin Manager (Диспетчер подключаемых модулей), то он не загружается.) Эту команду придется выполнить лишь один раз по ходу исполнения кода данного метода. После загрузки классов системных подключаемых модулей в рабочую область оперативной памяти можно инициировать одно или несколько системных событий.

Системные подключаемые модули, по существу, являются объявлениями классов, и поэтому выполнение кода в них не предполагается. А далее в методе render() из класса JSite фактически вызывается метод onAfterRender(), как показано ниже.

```
// инициировать событие onAfterRender
$this->triggerEvent('onAfterRender');
```

В этой строке кода инициируется событие onAfterRender. Рассмотрим подробнее, как это делается. Переменная $this ссылается на объект типа JSite, а следовательно,

в операторе `$this->triggerEvent` вызывается метод `triggerEvent()` из класса `JSite` с единственным аргументом в виде символьной строки `'onAfterRender'`. Класс `JSite` расширяет класс `JApplication` (из файла `libraries/joomla/application/application.php`). А поскольку в классе `JSite` отсутствует собственный метод `triggerEvent()`, т.е. он не переопределяет этот метод, наследуемый от родительского класса, то он вызывает данный метод из класса `JApplication`.

Итак, метод `triggerEvent()` из класса `JApplication` выполняется, а его код приведен ниже.

```
function triggerEvent($event, $args=null)
{
    $dispatcher = JDispatcher::getInstance();
    return $dispatcher->trigger($event, $args);
}
```

В коде данного метода сначала создается объект типа `JDispatcher`, а затем вызывается метод `trigger()`. Первый аргумент `$event` этого метода получает строковое значение `'onAfterRender'`, а второй аргумент `$args` — исходное пустое значение `null`, поскольку второй аргумент при вызове метода `triggerEvent()` не передается. В конечном итоге это приводит к выполнению метода `onAfterRender()` для каждого активизированного подключаемого модуля, доступного в рабочей области оперативной памяти. В данном случае поиск в рабочей области оперативной памяти ограничивается системными подключаемыми модулями, поскольку при вызове метода `importPlugin('system')` был указан лишь системный тип подключаемых модулей.

Как правило, имена подключаемых модулей должны быть согласованы с их типом. В частности, в системных подключаемых модулях должны использоваться имена только системных событий. Если следовать этому правилу, то уже не имеет никакого значения, загружены ли в оперативную память подключаемые модули других типов, поскольку выполняться будут лишь те из них, которые совпадают по типу с инициируемыми событиями.

Реализация метода запуска подключаемого модуля

Не вдаваясь особенно в подробности реализации метода `trigger()` в Joomla, заметим, что для создания подключаемых модулей нужно знать лишь основное назначение этого метода: выполнять все методы, совпадающие по имени с инициируемыми событиями. Если же вам интересно углубленно разобраться в принципе действия метода `trigger()`, проанализируйте его исходный код. В итоге вы обнаружите, что он разработан в Joomla на основе шаблона проектирования, действующего по принципу "наблюдателя", где события являются "наблюдаемыми", а подключаемые модули — "наблюдателями".

Что он делает

А теперь рассмотрим исходный код подключаемого модуля SEF. Ниже приведен фрагмент кода из первой части исходного файла `sef.php`.

```
// прямой доступ отсутствует
defined ('_JEXEC') or die;
```

```
/**
 * Подключаемый модуль SEF для Joomla!
 *
 * @package Joomla
 * @subpackage System
 */
class plgSystemSef extends JPlugin
{
    /**
     * Преобразование URL сайта в соответствии с
     * запросом по протоколу HTTP
     */
    public function onAfterRender()
    {
```

В первой строке приведенного выше фрагмента кода (эта строка следует после блока документации, который здесь не показан) выполняется стандартная команда `defined`, гарантирующая выполнение этого кода в среде Joomla. До появления версии Joomla 2.5 для импорта библиотеки из файла `libraries/joomla/plugin/plugin.php` требовалась команда `jimport`. А начиная с версии 2.5 этот файл автоматически загружается автозагрузчиком платформы Joomla. Он содержит класс `JPlugin`, являющийся родительским для всех подключаемых модулей. Он используется для объявления класса `plgSystemSef` как подкласса `JPlugin`. В данном случае подключаемый модуль относится к системному типу (`system`), а следовательно, его файл имеет полное имя `plgSystemSef`. И наконец, в рассматриваемом здесь фрагменте кода объявляется метод `onAfterRender()`.

Употребление прописных и строчных букв в именах методов и классов

В языке PHP не различаются прописные и строчные буквы в именах методов и классов. Но при их именовании принято пользоваться смешанным написанием, где первая буква каждого слова является прописной, а остальные — строчными. Как правило, имена классов начинаются с прописной буквы, тогда как имена методов — со строчной, но в именах подключаемых модулей первой указывается строчная буква. Такие условные обозначения упрощают чтение исходного кода.

Рассматриваемый здесь подключаемый модуль обнаруживает в HTML-документе ссылки и преобразует их в удобные для поискового механизма ссылки. Не вдаваясь особенно в подробности, обратим внимание лишь на две самые примечательные особенности метода `onAfterRender()`. Во-первых, рассмотрим следующий фрагмент кода в начале этого метода:

```
if ($app->getName() != 'site' || $app->getCfg('sef')=='0') {
    return true;
}
```

В этом фрагменте кода проверяются два условия. Первое условие, `$app->getName() != 'site'`, позволяет проверить, *не* находимся ли мы в пользовательской части веб-сайта,

а второе условие, `app->getCfg('sef')=='0'`, — установлен ли в нуль (т.е. выключен) параметр SEF, определяющий удобные для поиска URL в компоненте Global Configuration (Глобальная настройка). Если оба эти условия истинны, происходит немедленный возврат из данного метода вместе с логическим значением `true`.

Для чего нужны эти проверки? Мы можем изменить URL только при следующих двух условиях: мы находимся в пользовательской части веб-сайта и включен параметр SEF, определяющий удобные для поиска URL. Следует, однако, иметь в виду, что рассматриваемый здесь подключаемый модуль выполняется *всякий раз*, когда наступает событие `onAfterRender`, не зависимо от того, находимся ли мы в пользовательской или административной части веб-сайта, а также от установки параметра SEF. Упомянутые выше проверки приходится вводить в подключаемый модуль именно для того, чтобы убедиться в наличии условий для его выполнения. Сначала эти условия проверяются, и если они не выполняются, то происходит возврат из метода `onAfterRender()`, прежде чем в объект документа удастся внести какие-нибудь изменения.

И во-вторых, метод `onAfterRender()` не принимает никаких аргументов и возвращает логическое значение, обозначающее, насколько успешно он был выполнен. Для разных типов подключаемых модулей предусмотрены разные сигнатуры данного метода (т.е. наборы передаваемых ему аргументов) и возвращаемые значения, поэтому их следует непременно учитывать при создании собственного подключаемого модуля.

Подключаемый модуль аутентификации

Этот подключаемый модуль выполняется при входе пользователя на веб-сайт. Он проверяет достоверность имени и пароля пользователя. Это один из трех подключаемых модулей аутентификации, включенных в основной вариант распространения Joomla, и он реализует способ, используемый по умолчанию для проверки пользователей веб-сайта Joomla. Рассматриваемый здесь подключаемый модуль находится в файле `plugins/authentication/joomla/joomla.php`, а имя его класса — `plgAuthenticationJoomla`.

Как он запускается на выполнение

При попытке пользователя войти на веб-сайт Joomla вызывается на выполнение метод `authenticate()` из класса `JAuthentication`, находящегося в файле `libraries/joomla/user/authentication.php`. Как и предполагалось, в этом методе присутствует следующая строка кода:

```
$plugins = JPluginHelper::getPlugin('authentication');
```

В этой строке кода все подключаемые модули аутентификации загружаются в рабочую область памяти. Далее в данном методе выполняется следующий цикл `foreach`:

```
foreach ($plugins as $plugin)
{
    $className = 'plg'.$plugin->type.$plugin->name;
    if (class_exists($className)) {
       $plugin = new $className($this, (array)$plugin);
    }
    else {
      // выйти, если нельзя создать объект подключаемого модуля
      JError::raiseWarning(50, JText::sprintf(
```

```
↳'JLIB_USER_ERROR_AUTHENTICATION_FAILED_LOAD_PLUGIN', $className));
      continue;
    }

    // попытаться произвести аутентификацию
    $plugin->onUserAuthenticate($credentials, $options, $response);
```

В приведенном выше цикле любые активизированные подключаемые модули проверяются по очереди на наличие в них классов с заданным именем. Если активизированный подключаемый модуль отсутствует, цикл завершается выдачей соответствующего сообщения об ошибке. А если все классы существуют, выполняется последняя строка кода, в которой для каждого подключаемого модуля запускается на выполнение метод `onUserAuthenticate()`. Обратите внимание на то, что этому методу передаются три аргумента: `$credentials`, `$options`, `$response`. Более подробно они рассматриваются в следующем подразделе.

Что он делает

Исходный код метода `onUserAuthenticate()` приведен в листинге 5.1.

Листинг 5.1. Метод `onUserAuthenticate()` для аутентификации средствами Joomla

```
function onUserAuthenticate($credentials, $options, &$response)
{
    $response->type = 'Joomla';
    // В Joomla! пустые пароли не допускаются
    if (empty($credentials['password'])) {
      $response->status = JAUTHENTICATE_STATUS_FAILURE;
      $response->error_message =
↳Text::_('JGLOBAL_AUTH_EMPTY_PASS_NOT_ALLOWED');
      return false;
    }

    // инициализировать переменные
    $conditions = '';

    // получить объект базы данных
    $db = JFactory::getDbo();
    $query = $db->getQuery(true);

    $query->select('id, password');
    $query->from('#__users');
    $query->where('username=' . $db->Quote($credentials['username']));

    $db->setQuery($query);
    $result = $db->loadObject();

    if ($result) {
      $parts = explode(':', $result->password);
      $crypt = $parts[0];
      $salt = @$parts[1];
      $testcrypt = JUserHelper::getCryptedPassword(
        $credentials['password'], $salt);
```

```
      if ($crypt == $testcrypt) {
        //  привести в соответствие с остальной системой
        $user = JUser::getInstance($result- >id);
        $response->email = $user->email;
        $response->fullname = $user->name;
        if (JFactory::getApplication()->isAdmin()) {
          $response->language = $user->getParam('admin_language');
        }
        else {
          $response->language = $user->getParam('language');
        }
        $response->status = JAUTHENTICATE_STATUS_SUCCESS;
        $response->error_message = '';
      } else {
        $response->status = JAUTHENTICATE_STATUS_FAILURE;
        $response->error_message =
↳JText::_('JGLOBAL_AUTH_INVALID_PASS');
      }
    } else {
      $response->status = JAUTHENTICATE_STATUS_FAILURE;
      $response->error_message = JText::_('JGLOBAL_AUTH_NO_USER');
    }
}
```

Проанализируем исходный код данного метода, начав с двух ниже приведенных первых строк.

```
function onUserAuthenticate($credentials, $options, &$response)
{
```

Метод `onUserAuthenticate()` принимает три аргумента. Первый аргумент `$credentials` представляет собой ассоциативный массив с двумя элементами: `"password"` и `"username"`, обозначающими соответственно пароль и имя пользователя, введенные в форме регистрации. Второй аргумент `$options` в данном методе не используется. А третий аргумент `$response` имеет особое значение. Обратите внимание на предшествующий ему знак амперсанда (`&`), указывающий на то, что этот аргумент передается по ссылке. Это означает, что при внесении изменений в объект во время выполнения вызываемого метода они будут доступны и в вызывающем методе. Подробнее об этом — в приведенной далее врезке "Присваивание и передача объектов по ссылке".

Этот метод возвращает логическое значение `false`, если регистрация не прошла успешно, а иначе он завершается, не возвращая никакого значения. Но данные вызывающему методу все же передаются, поскольку в объект `$response` внесены изменения, которые становятся доступными вызывающему методу по завершении данного метода.

Присваивание и передача объектов по ссылке

Для версии Joomla 2.5 требуется PHP 5.2 или более поздняя версия этого языка, тогда как версия Joomla 1.6 могла вполне работать и с PHP 4.0. В версии PHP 5 было изменено исходное поведение объектов, когда они присваиваются переменным. А в версии 4.0 при наличии

переменной ссылки на объект, например, `$myObject`, выполнялась следующая операция присваивания:

```
$x = $myObject;
```

Переменная `$x` создавалась как копия переменной `$myObject`. Поэтому если переменная `$myObject` изменялась далее в коде, это не оказывало никакого влияния на переменную `$x`. Но если выполнялась следующая операция присваивания:

```
$x = &$myObject;
```

то знак `&` указывал на необходимость создать в коде PHP переменную `$x` в качестве еще одной ссылки на переменную `$myObject`. Такая операция называется присваиванием по ссылке, и в данном примере обе переменные, `$x` и `$myObject`, ссылаются на один и тот же объект. Если впоследствии изменится переменная `$myObject`, это отразится и на переменной `$x`, поскольку обе они, по существу, обозначают один и тот же объект.

Это же справедливо и для ссылок на объекты в сигнатурах методов, как показано в приведенном ниже примере.

```
function onUserAuthenticate($credentials, $options, &$response)
```

В данном случае знак `&` указывает на то, что методу передается ссылка на объект `$response`. Если переменная `$response` изменяется в ходе выполнения метода, то фактически изменяется один и тот же объект, передаваемый по ссылке. Это означает, что при выходе из вызываемого метода и возврате в вызывающий метод любые изменения в переменной `$response` отразятся на состоянии той же самой переменной в вызывающем методе.

С версии PHP 5 необходимость указывать знак `&` в первом рассмотренном здесь примере отпала. Так, при выполнении следующей операции:

```
$x = $myObject;
```

присваивание в PHP 5 выполняется по ссылке. Если же требуется создать новый объект, для этого необходимо выполнить следующую команду:

```
$x = clone $myObject;
```

и в результате создается копия объекта.

Если проанализировать исходный код версии Joomla 1.5, то во многих его местах можно обнаружить операторы присваивания по ссылке со знаком `&`, как того требовала спецификация PHP 4.0. А в версии Joomla 2.5 эти знаки были удалены, поскольку необходимость в них отпала.

Совсем иначе обстоит дело с передачей объектов по ссылке в сигнатурах методов. В этом случае рекомендуется по-прежнему пользоваться знаком `&`. Во-первых, этот знак указывает на то, что любые изменения в объекте во время выполнения вызываемого метода будут доступны и в вызывающем методе. И во-вторых, в силу упомянутых выше отличий в версиях языка PHP знак `&` явно указывает на передачу объекта по ссылке.

До появления версии Joomla 2.5 для импорта пользовательского вспомогательного класса требовалась команда `jimport`. Этот класс требуется для последующего шифрования проверочного пароля. Начиная с версии Joomla 2.5 этот класс загружается автозагрузчиком.

Перейдем к анализу следующей строки кода:

```
$response->type = 'Joomla';
```

В этой строке кода устанавливается строковое значение 'Joomla' в поле type объекта $response. Это поле служит для указания на тот подключаемый модуль аутентификации, который использовался для проверки подлинности пользователя.

Далее следует приведенный ниже кодовый блок.

```
// В Joomla! пустые пароли не допускаются
if (empty($credentials['password'])) {
    $response->status = JAUTHENTICATE_STATUS_FAILURE;
    $response->error_message = JText::_(
↳'JGLOBAL_AUTH_EMPTY_PASS_NOT_ALLOWED');
    return false;
}
```

Этот кодовый блок содержит условный оператор if для проверки ввода пароля. Если пароль не введен, произвести аутентификацию не удастся. Для указания на это устанавливаются соответствующие значения в полях status и error_message объекта $response, а из рассматриваемого здесь метода возвращается логическое значение false.

В следующем кодовом блоке выполняется простой запрос базы данных Joomla с целью получить из нее идентификатор пользователя и пароль. Это первый пример запроса базы данных, в котором используется класс JDatabaseQuery, внедренный в версии Joomla 1.6. В строке кода

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

создается объекта типа JDatabase. Как правило, это первая стадия формирования любого запроса базы данных. В следующей строке кода

```
$query = $db->getQuery(true);
```

создается объект типа JDatabaseQuery. А в следующей далее строке кода

```
$query->select('id, password');
```

в оператор SELECT запроса вводятся идентификатор столбцов таблицы в базе данных и пароль. Далее следует строка кода, где в запрос вводится таблица #__users:

```
$query->from('#__users');
```

Обратите внимание на то, что доступ к таблице осуществляется с префиксом #__ в виде знака денежной единицы и двух последующих символов подчеркивания. До выполнения запроса этот префикс заменяется префиксом таблицы, выбираемым при установке Joomla (например, префиксом jos_).

В следующей строке кода

```
$query->where('username=' . $db->Quote($credentials['username']));
```

вводится оператор WHERE, ограничивающий запрос теми строками таблицы, где столбец username совпадает с элементом username массива $credentials. А поскольку столбец username должен быть единственным в таблице базы данных, то по данному запросу будет получена лишь одна строка.

Метод $db->Quote() играет очень важную роль в обеспечении безопасности. Значение его аргумента username заключается в одинарные кавычки и "экранирует" любые символы, имеющие особое значение в запросах SQL, например, в тех случаях, если одинарные или двойные кавычки завершают один оператор SQL и начинают другой.

Во избежание подобных недоразумений они преобразуются в управляющие последовательности символов \\' и \\". В итоге специальное значение этих символов игнорируется в базе данных, защищая от случайного их ввода в поле имени пользователя, а следовательно, и в команде SQL.



Предупреждение в целях безопасности: пользуйтесь методом $db->Quote() и операторами (int) и (float) во избежание умышленного внесения запросов SQL

В рассматриваемом здесь примере аргумент `$credentials['username']` метода `$db->Quote()` введен пользователем веб-сайта. Как пояснялось ранее, вводимые данные должны быть защищены от попыток злоумышленников ввести злонамеренные команды SQL в полях ввода данных. Подобного рода хакерская атака или попытка взлома называется умышленным внесением запросов SQL.

Предотвратить умышленное внесение запросов SQL можно, соблюдая два простых правила.

1. Если вводимое значение предполагается целочисленным (например, **-1** или **1234**) или с плавающей точкой либо запятой (вроде **12.3456** или **-2,3**), воспользуйтесь операторами `(int)` и `(float)` языка PHP для преобразования (или приведения) этого значения к нужному типу. Например, в следующей строке кода

   ```
   $query->where('id =' . (int) $id);
   ```

   оператор `(int)` выполняет преобразование исходного значения аргумента `$id` в целое. Этим гарантируется, что в команде формируемого запроса SQL не появится ничего, кроме целочисленного значения и знака "минус", а все остальное будет отброшено. Благодаря этому исключается возможность ввести любые команды SQL в переменных и аргументах методов.

2. Для указания переменных и аргументов любых типов, кроме целочисленных и с плавающей точкой (например, текста и дат), пользуйтесь методом `$db->Quote()`, чтобы гарантировать надежность значений, указываемых в запросе, как показано в приведенном ниже примере.

   ```
   $query->where('title =' . $db->quote($myTitle);
   ```

3. Если аргумент `$myTitle` содержит кавычки или другие символы, имеющие особое значение в командах SQL, они экранируются. В итоге их особое значение будет проигнорировано в базе данных, где они будут интерпретированы как обычный текст.

Следуя приведенным выше правилам, вы сможете защитить свои данные и воспрепятствовать попыткам злоумышленников выполнить несанкционированные запросы.

Итак, запрос базы данных сформирован и все готово для его выполнения. Это делается в следующем фрагменте кода:

```
$db->setQuery($query);
$result = $db->loadObject();
```

В первой строке приведенного выше фрагмента кода запрос передается объекту базы данных, тогда как во второй строке — выполняется сам запрос базы данных, а результаты сохраняются в переменной `$result`. Если по какой-нибудь причине выполнение запроса завершится неудачно, переменная `$result` окажется пустой или с логическим значением `false`.

Остальная часть рассматриваемого здесь метода представляет собой кодовый блок условных операторов `if/then/else`, начинающийся следующим образом:

```
if ($result) {
    $parts = explode(':', $result->password);
    $crypt = $parts[0];
    $salt = @$parts[1];
    $testcrypt = JUserHelper::getCryptedPassword($credentials['password'],
↳$salt);
```

В первой строке кода переменная `$result` проверяется на наличие логического значения `true`. Если она не содержит это логическое значение, осуществляется переход к кодовому блоку внешнего оператора `else`, как показано ниже.

```
} else {
    $response->status = JAUTHENTICATE_STATUS_FAILURE;
    $response->error_message = JText::_('JGLOBAL_AUTH_NO_USER');
}
```

В итоге пользователь получает сообщение об ошибке, уведомляющее о том, что его регистрация на веб-сайте не состоялась.

Применение нелогических значений в условных операторах `if` языка PHP

Начинающим программировать на PHP применение нелогических значений переменных, отличающихся от `true` или `false`, в условных операторах `if` может показаться не совсем понятным. Например, переменная `$result` в операторе `if ($result)` совсем не обязательно должна содержать логическое значение. И если это значение не является логическим, то оно сначала преобразуется в логическое значение, а затем вычисляется. Когда переменная `$result` содержит любые данные, кроме нулевого или пустого значения, ее содержимое определяется как логическое значение `true`, и поэтому далее выполняется кодовый блок, следующий после условного оператора `if`. Так, если переменная `$result` содержит объект или массив, ее содержимое определяется как логическое значение `true` независимо от хранящихся в ней данных.

Например, присваивание `$result = $db->loadObject()` вполне допустимо, поскольку метод `loadObject()` возвращает объект или логическое значение `false`. Если он возвращает объект, то последний всегда определяется как логическое значение `true` в условном операторе `if`. Таким образом, мы можем выяснить, получен ли достоверный результат запроса базы данных или ошибка.

Обрабатывая нулевые или пустые значения, следует, однако, не терять бдительность, чтобы не попасть в следующую западню. Если, например, метод может возвратить нулевое значение как достоверное, но возвращает логическое значение `false`, указывающее на возникновение ошибки, то код `if (!$result)` не сможет проверить наличие этой ошибки. Когда возвращается нулевое значение, содержимое переменной `$result` определяется как логическое значение `false`, и поэтому результат обратного выражения `!$result` вычисляется как логическое значение `true`. И следующий далее кодовый блок будет выполняться, как будто никакой ошибки не произошло, хотя на самом деле она возникла.

Аналогичным образом неправильно сработает условный оператор `if ($result == false)`. И хотя это не так очевидно, тем не менее программная ошибка в данном примере оказывается такой же, как и в предыдущем. Если переменная `$result` содержит нулевое значение, ее содержимое определяется как логическое значение `false`, и поэтому результат выражения `$result`

`== false` вычисляется как логическое значение `true`. И следующий далее кодовый блок будет выполняться, как будто никакой ошибки не произошло.

В качестве выхода из этого положения рекомендуется пользоваться оператором `===` (с тремя знаками равенства) языка PHP для проверки на точность совпадения. Это означает, что оба сравниваемых элемента должны быть одного и того же *типа* (логического, целочисленного, строкового и т.д.), а также должны иметь одинаковое *значение*. Следовательно, результат выражения `$result === false` вычисляется как логическое значение `true` только в том случае, если переменная `$result` содержит логическое значение `false`. Применительно к рассматриваемому здесь примеру программная ошибка устраняется, даже если переменная `$result` содержит нулевое значение. В этом случае результат выражения `$result === false` вычисляется как логическое значение `false`.

Операторы `===` и `!==` осуществляют проверку на точность совпадения сравниваемых элементов по типу и значению, и поэтому ими удобно пользоваться при возврате из методов нулевых целочисленных значений, а также пустых (`" "`) и нулевых (`"0"`) строковых значений в качестве достоверных результатов.

Если после выполнения запроса базы данных возвращается достоверный результат (в переменной `$result`), выполняется кодовый блок условного оператора `if`. Первая часть этого блока приведена ниже.

```
$parts = explode(':', $result->password);
$crypt = $parts[0];
$salt = @$parts[1];
$testcrypt = JUserHelper::getCryptedPassword($credentials['password'],
↳$salt);
```

Пароль хранится в двух полях базы данных Joomla, разделяемых двоеточием. В первой строке приведенного выше фрагмента кода используется функция `explode()` языка PHP для размещения двух частей столбца таблицы с паролем в массиве `$parts`. Затем первая часть пароля сохраняется в переменной `$crypt`, а вторая его часть — в переменной `$salt`.

По умолчанию для шифрования паролей в Joomla применяется алгоритм одностороннего хеширования под названием `md5`. Под термином "одностороннее" подразумевается только шифрование пароля, но не его дешифрирование. Поэтому для проверки правильности введенного пароля значение, введенное пользователем, шифруется и сохраняется в переменной `$testcrypt`.

Далее следует очередной кодовый блок условных операторов `if/then/else`, в котором проверяется, совпадает ли зашифрованное значение введенного пароля с зашифрованным значением, хранящимся в базе данных. Этот кодовый блок приведен ниже.

```
if ($crypt == $testcrypt) {
    // привести в соответствие с остальной системой
    $user = JUser::getInstance($result- >id);
    $response->email = $user->email;
    $response->fullname = $user->name;
    if (JFactory::getApplication()->isAdmin()) {
      $response->language = $user->getParam('admin_language');
    }
    else {
```

```
    $response->language = $user->getParam('language');
  }
  $response->status = JAUTHENTICATE_STATUS_SUCCESS;
  $response->error_message = '';
  } else {
    $response->status = JAUTHENTICATE_STATUS_FAILURE;
    $response->error_message = JText::_('JGLOBAL_AUTH_INVALID_PASS');
  }
```

В первой части приведенного выше фрагмента кода проверяется правильность введенного пароля. Для этого получается объект пользователя, и в полях `email` и `fullname` объекта `$response` устанавливаются значения из соответствующих полей объекта пользователя. Затем получается объект подходящего языка в зависимости от конкретной части веб-сайта (пользовательской или административной). И наконец, в поле `status` объекта `$response` устанавливается состояние успешной регистрации. Если же сравниваемые пароли не совпадают, то в поле `status` устанавливается состояние неудачной регистрации, а в поле `error_message` — соответствующее сообщение об ошибке.

Обратите внимание на то, что команда возврата не выдается при успешной регистрации. Вместо этого в рассматриваемом здесь методе применяется специальный прием для восстановления связи с вызывающим методом. Этот прием состоит в том, что переменная `$response` изменяется в данном методе, чтобы раскрыть значения полей из объекта достоверного пользователя.

Напомним, что в рассматриваемом здесь примере метод `authenticate()` подключаемого модуля был вызван из метода `authenticate()` класса `JAuthentication`. Рассматривая исходный код класса `JAuthentication` после вызова метода `authenticate()` из подключаемого модуля, можно обнаружить следующий фрагмент кода:

```
// выйти из цикла, если аутентификация прошла успешно
if ($response->status === JAUTHENTICATE_STATUS_SUCCESS)
{
    if (empty($response->type)) {
        $response->type = isset($plugin->_name) ? $plugin->_name :
          $plugin->name;
    }

    if (empty($response->username)) {
        $response->username = $credentials['username'];
    }

    if (empty($response->fullname)) {
        $response->fullname = $credentials['username'];
    }

    if (empty($response->password)) {
        $response->password = $credentials['password'];
    }
  }
}
return $response;
```

Этот видоизмененный вариант объекта $response доступен в рассматриваемом здесь методе и фактически возвращается им. И хотя метод authenticate() подключаемого модуля не возвращает сам объект $response, он передает обратно вызывающему методу результаты своего выполнения через обновленный объект $response.

Подключаемый модуль содержимого

Этот подключаемый модуль находится в файле plugins/content/joomla/joomla.php, а имя его класса — plgContentJoomla. У него имеются два метода. В частности, метод onContentAfterSave() служит для отправки пользователям уведомления по электронной почте после сохранения статьи, а метод onContentBeforeDelete() — для проверки наличия в категории содержимого любых элементов, присвоенных ей перед ее удалением. Рассмотрим подробнее метод onContentBeforeDelete().

Как он запускается на выполнение

Когда пользователь удаляет категории, статьи, контактную информацию и прочие элементы содержимого в административной части веб-сайта на Joomla, инициируется событие onContentBeforeDelete. Это, в частности, делается в классе JModelAdmin, находящемся в файле libraries/joomla/application/component/modeladmin.php. Если проанализировать исходный код метода delete() из этого класса, то в нем можно обнаружить следующий фрагмент кода:

```
// инициировать событие
$result = $dispatcher->trigger($this->event_before_delete,
↳array($context, $table));
```

В поле event_before_delete конструктора класса JModelAdmin устанавливается строковое значение onContentBeforeDelete. В отношении приведенного выше фрагмента кода необходимо отметить следующее. Во-первых, предполагается, что значение, возвращаемое из данного метода, сохраняется в переменной $result. И во-вторых, методу trigger() передаются два аргумента: имя события и массив, состоящий из двух элементов. Метод trigger() распаковывает массив и передает каждый его элемент в качестве аргументов методу onContentBeforeDelete(). В данном случае двумя аргументами являются переменные $context и $table. В частности, переменная $context предназначена для сообщения некоторых сведений о том контексте, в котором данное событие инициировано (например, в контексте категорий "com_categories.category"). А переменная $table представляет собой массив удаляемых данных.

Что он делает

В первой части метода onContentBeforeDelete() находится следующий фрагмент кода:

```
public function onContentBeforeDelete($context, $data)
{
    // пропустить подключаемый модуль, если удаляется
    // нечто иное, чем категории
    if ($context != 'com_categories.category') {
        return true;
}
```

Как пояснялось ранее, этот подключаемый модуль будет выполняться в любой момент, когда пользователь удаляет какое угодно содержимое. Но поскольку в этом подключаемом модуле проверяется наличие любых элементов, присвоенных категории, то его выполнение имеет смысл лишь в контексте удаления категории. Поэтому в первом условном операторе `if` проверяется, действительно ли удаляется категория, а не что-нибудь другое. Если это не категория, то происходит возврат из рассматриваемого здесь метода с логическим значением `true`.

Как упоминалось ранее, у каждого подключаемого модуля имеется свой файл исходного кода на языке PHP и соответствующий XML-файл. В частности, XML-файл подключаемого модуля выполняет следующие три функции. Во-первых, в нем предоставляются описательные сведения о подключаемом модуле, в том числе его имя, версия, дата создания, автор и лицензионное соглашение. Во-вторых, в нем перечисляются все файлы, которые требуется установить или удалить. И в-третьих, в нем определяются любые параметры или режимы, которые могут быть установлены при использовании подключаемого модуля. Эти параметры и режимы оказываются доступными на экране Plugin Manager, когда подключаемый модуль открывается для редактирования.

Параметры в Joomla позволяют администратору настроить режим работы веб-сайта, не прибегая к программированию. В данном примере подключаемый модуль содержимого (`Content→Joomla!`) дает администратору возможность указать, следует ли проверять категории на предмет их пустоты перед удалением. И делается это с помощью параметра `check_categories`. Более подробно параметры настройки подключаемых модулей рассматриваются далее в этой главе.

Параметр `check_categories` позволяет администратору отменить проверку категорий. И делается это в следующем фрагменте кода из рассматриваемого здесь метода:

```
// проверить, активизирована ли данная функция
if (!$this->params->def('check_categories', 1)) {
    return true;
}
```

В приведенном выше фрагменте кода `$this->params` — это объект класса `JRegistry`, содержащий параметры, сохраняемые в таблице `#__extensions` базы данных для данного подключаемого модуля. Метод `def()` считывает значение этого параметра или использует значение **1**, если параметр не определен. Напомним, что в языке PHP нулевое значение определяется как логическое значение `false`. Именно это обстоятельство используется в данном фрагменте кода.

Параметр `check_categories` должен иметь нулевое значение, если проверять категории не нужно, а иначе — единичное значение. Так, если он имеет нулевое значение, условие, проверяемое с помощью оператора отрицания `!`, окажется истинным, а следовательно, выполнение метода преждевременно завершится возвратом логического значения `true`.

Далее в рассматриваемом здесь методе следует приведенная ниже строка кода.

```
$extension = JRequest::getString('extension');
```

В этой строке кода переменная `$extension` получает значение переменной из массива `$_REQUEST` в коде PHP. Обратите внимание на то, что для этой цели используется библиотечный метод `JRequest::getString()`. Можно было бы, конечно, прочитать содержимое массива `$_REQUEST` непосредственно, но для этой цели настоятельно

рекомендуется использоваться методами из класса JRequest, поскольку они выполняют встроенную фильтрацию данных. В данном случае метод getString() отфильтровывает символы в шестнадцатеричном коде, а также символы, кодированные в формате URL. В классе JRequest предоставляется целый ряд методов для чтения переменных запроса (getString(), getInt(), getWord() и т.д.), причем использовать всегда следует наиболее строгий на ограничения типов метод. Иными словами, если заранее известно, что значение в запросе должно всегда быть целочисленным, то для его чтения следует вызвать метод getInt(). Все имеющиеся типы фильтров приведены в приложении Б.

Ниже приведен следующий фрагмент кода из рассматриваемого здесь метода.

```
// истина по умолчанию, если это не основное расширение
$result = true;

$tableInfo = array (
    'com_banners' => array('table_name' => '#__banners'),
    'com_contact' => array('table_name' => '#__contact_details'),
    'com_content' => array('table_name' => '#__content'),
    'com_newsfeeds' => array('table_name' => '#__newsfeeds'),
    'com_weblinks' => array('table_name' => '#__weblinks')
);
```

В начале этого фрагмента кода переменной $result по умолчанию присваивается логическое значение true. Затем создается массив из имен различных таблиц для разных типов расширений. Рассматриваемый здесь подключаемый модуль будет пригодным лишь для указанных выше пяти типов расширений. В данном массиве для каждого типа расширения указывается имя отдельной таблицы. Ниже приведен следующий фрагмент кода из рассматриваемого здесь метода.

```
// а теперь проверить, известно ли данное расширение как основное
if (isset($tableInfo[$extension]))
{
    // получить имя таблицы для известного основного расширения
    $table = $tableInfo[$extension]['table_name'];
    // проверить, имеются ли в данной категории
    // какие-нибудь элементы содержимого
    $count = $this->_countItemsInCategory($table, $data->get('id'));
```

В этом фрагменте кода проверяется, находится ли текущее расширение в массиве из пяти основных расширений. Если оно там находится, то выполняется код в условном операторе if. А если текущее расширение не относится к числу пяти основных расширений, то происходит переход в самый конец рассматриваемого здесь метода и возврат из него значения переменной $result, которой раньше было присвоено логическое значение true.

В кодовом блоке условного оператора if переменной $table сначала присваивается определенное ранее имя таблицы. Затем в переменной $count устанавливается значение, возвращаемое закрытым методом _countItemsInCategory(). Этот метод выполняет запрос базы данных, чтобы определить, сколько элементов (статей, контактной информации и т.д.) имеется в данной категории. Обратите внимание на то, что в качестве аргументов этому методу передаются имя таблицы ($table) и значение, возвращаемое методом $data->get('id'), т.е. значение, выбираемое из поля id объекта $data для данной категории.

Имена и область действия переменных в методах

У начинающих программировать может возникнуть недоумение по поводу переменной `$table`. Напомним, что в коде, инициирующем событие `onContentBeforeDelete`, передается массив, определяемый как `array($context, $table)`, и что этот массив распаковывается, превращаясь в два аргумента метода `onContentBeforeDelete()`.

Второй передаваемый аргумент назван `$table`. Но ведь второй аргумент в сигнатуре метода `onContentBeforeDelete()` называется `$data`. Несмотря на отличие в именах, переменная `$data` в данном методе имеет такое же значение, какое было у переменной `$table`, когда инициировалось событие.

Когда методам и функциям передаются аргументы, имеет значение *расположение*, а не *название* аргумента. Первая переменная из вызывающего метода загружается в первый аргумент сигнатуры вызываемого метода и так далее. Имя аргумента в сигнатуре метода соответствует имени переменной в теле этого метода.

В программировании это понятие называется *областью действия*. К области действия переменной относится часть программы, где эта переменная имеет конкретное значение. В языке PHP большинство переменных являются локальными по своей области действия. Это означает, что они определены только в теле того метода или функции, где применяются. Замечательная особенность локальных переменных состоит в следующем: программирующему не нужно беспокоиться о том, что точно такое же имя переменной может быть использовано в каком-нибудь другом месте программы. Ему нужно лишь отслеживать имена переменных в теле одного метода.

Переменная `$table` является локальной для метода `onContentBeforeDelete()`, а следовательно, она может иметь несколько иное значение в этом методе, чем в каком-нибудь другом месте программы. Но поскольку имя переменной `$table` не указывается в сигнатуре метода `onContentBeforeDelete()`, то ничто не мешает воспользоваться им в теле этого метода с какой угодно целью. В данном случае переменная `$table` ссылается на имя таблицы, определяемое в массиве `$tableInfo`.

Перейдем к следующему фрагменту кода из рассматриваемого здесь метода.

```
// возвратить логическое значение false, если произойдет
// ошибка при обращении к базе данных
if ($count === false)
{
    $result = false;
}
```

В этом фрагменте кода проверяется, получен ли достоверный результат из метода `_countItemsInCategory()`. Этот метод возвращает числовое или логическое значение `false`. Обратите внимание на применение оператора `===` при проверке наличия в переменной `$count` логического значения `false`. Это приходится делать потому, что нулевое значение, возвращаемое из метода `_countItemsInCategory()`, является достоверным результатом. Если этот метод возвратит логическое значение `false`, значит, при выполнении запроса базы данных по какой-то причине произошла ошибка. А если он возвратит достоверный результат, то произойдет переход к приведенному ниже кодовому блоку `else`.

```
else
  {
    // сообщить об ошибке, если в категории найдены элементы содержимого
    if ($count > 0) {
    $msg = JText::sprintf('COM_CATEGORIES_DELETE_NOT_ALLOWED',
↳$data->get('title')) .
    JText::plural('COM_CATEGORIES_N_ITEMS_ASSIGNED', $count);
    JError::raiseWarning(403, $msg);
    $result = false;
  }
  // проверить наличие элементов в любых порожденных категориях,
  // а если это лист дерева, то порожденные категории отсутствуют
  if (!$data->isLeaf()) {
    $count = $this->_countItemsInChildren($table, $data->get('id'), $data);
    if ($count === false)
    {
      $result = false;
    }
    elseif ($count > 0)
    {
      $msg = JText::sprintf('COM_CATEGORIES_DELETE_NOT_ALLOWED',
↳$data->get('title')) .
      JText::plural('COM_CATEGORIES_HAS_SUBCATEGORY_ITEMS', $count);
      JError::raiseWarning(403, $msg);
      $result = false;
    }
  }
}
```

В первом условном операторе `if` из приведенного выше фрагмента кода проверяется, содержит ли переменная `$count` значение больше нуля. В этом случае пользователю выдается предупреждающее сообщение, а в переменной `$result` устанавливается логическое значение `false`. Следует иметь в виду, что, возвращая логическое значение `false`, рассматриваемый здесь подключаемый модуль предотвращает случайное удаление пользователем категории. Кроме того, оператор `return` употребляется лишь в самом конце данного метода, а следовательно, код продолжает выполняться.

В следующем условном операторе `if` проверяется, имеются ли в порожденных категориях какие-нибудь элементы содержимого, для чего вызывается метод `_countItemsInCategory()`. Обратите внимание на сокращенную запись, применяемую с целью сэкономить немного время обработки. У объекта `$data` имеется метод `isLeaf()`, возвращающий логическое значение `true`, если текущая категория является "листом" в "дереве" категорий, а это означает, что у нее нет никаких порожденных категорий. И в этом случае весь кодовый блок условного оператора `if` пропускается.

Если же имеются порожденные категории, а в них — какие-нибудь элементы содержимого, то выдается еще одно предупреждающее сообщение, а в переменной `$result` устанавливается логическое значение `false`. Таким образом, если выполняются оба проверяемых условия, то выдаются оба предупреждающих сообщения, а это означает, что элементы содержимого присутствуют как в самой текущей категории, так и в порожденных категориях. Для выдачи предупреждающего сообщения пользователю служит метод `JError::raiseWarning()`, причем в само сообщение включается подсчитанное количество элементов содержимого.

В версии Joomla 1.6 в класс `JText` был внедрен новый эффективный метод `plural()`, позволяющий автоматически выбирать подходящее языковое окончание в зависимости от того, какое число отображается: **1** или больше 1. Этот метод используется при выдаче обоих предупреждающих сообщений, чтобы написать в них слово "элемент" или "элементов", исходя из количества обнаруженных элементов содержимого, но не прибегая к условному оператору `if`. Кроме того, метод `JText::plural()` правильно выбирает различные формы слова для одного, двух и трех элементов на разных языках.

В конце рассматриваемого здесь метода употребляется оператор `return $result;`, который просто возвращает логическое значение `true`, если элементы содержимого в категории отсутствуют, а иначе — логическое значение `false`. Как упоминалось ранее, данный метод делает нечто полезное лишь в том случае, если из таблицы `#__categories` удаляются элементы. Но ведь мы могли бы с тем же успехом ввести аналогичную проверку в класс таблицы категорий. Зачем же вместо этого пользоваться подключаемым модулем?

Это делается ради удобства. Организовав подобную проверку в подключаемом модуле, мы предоставляем администратору веб-сайта целый ряд дополнительных возможностей. Во-первых, подключаемые модули могут быть выключены из работы, что дает администратору веб-сайта возможность удалить проверку категорий. Во-вторых, параметры подключаемого модуля позволяют включать или отключать проверки по отдельности. В-третьих, можно предоставить собственный подключаемый модуль, заменяющий или дополняющий функциональные возможности данного или любого другого основного подключаемого модуля. И наконец, данный подключаемый модуль предоставляет простую модель, которой могут воспользоваться сторонние разработчики расширений для организации в них проверки категорий на наличие элементов содержимого.

Рассмотренные выше примеры наглядно демонстрируют истинный потенциал подключаемых модулей в отношении повышения гибкости системы в целом. Их можно выключать из работы или заменять, не затрагивая основные файлы. А это дает больше возможностей для управления процессом обработки данных, подспудно выполняющейся в Joomla.

Подключаемый модуль для обработки события `onBeforeCompileHead`

А теперь перейдем к самому интересному. Напишем простой подключаемый модуль для обработки события `onBeforeCompileHead`. Это событие позволяет видоизменить элемент заголовка HTML-страницы перед ее воспроизведением. Следовательно, используя этот подключаемый модуль, мы можем видоизменить любой элемент, который предполагается использовать в заголовке HTML-документа, включая метаданные, заглавие, ссылку или элементы сценария.

Как он запускается на выполнение

Событие `onBeforeCompileHead` инициируется в методе `fetchHead()` класса `JDocumentRendererHtml` (из файла `libraries/joomla/document/html/renderer/head.php`). Этот метод считывает данные для элемента HTML-заголовка из объекта документа и выводит его в буфер непосредственно в текстовом формате HTML. Ниже приведен код, инициирующий данное событие.

```
// инициировать событие onBeforeCompileHead
$app = JFactory::getApplication();
$app->triggerEvent('onBeforeCompileHead');
```

Что он делает

Если проанализировать исходный код начальной HTML-страницы веб-сайта с установленными выборочными данными, то в элементе заголовка можно обнаружить следующие элементы метаданных:

```
<meta name="robots" content="index, follow" />
<meta name="keywords" content="My keywords." />
<meta name="rights" content="My rights." />
<meta name="language" content="en-GB" />
```

Разрабатываемый нами подключаемый модуль будет очень простым. Он должен ввести атрибут "исправлено" в элемент метаданных HTML-документа, исходя из значения параметра, вводимого пользователем для настройки данного подключаемого модуля. В частности, он должен выдать следующий результат:

```
<meta name="revised" content="Mark Dexter, 17 March 2012" />
```

где атрибут `content` содержит текст, введенный в качестве значения параметра данного подключаемого модуля.

Для этого мы должны разобраться, каким образом в объекте типа `JDocumentHTML` хранятся данные элемента HTML-заголовка. Итак, проведем небольшое исследование. В сигнатуре метода `fetchHead()` из класса `JDocumentRendererHead`, где инициируется событие `onBeforeCompileHead`, мы обнаруживаем переменную `$document`. Она содержит ссылку на объект типа `JDocumentHTML`, у которого имеется метод `getHeadData()`, возвращающий данные заголовка для документа или страницы. Если ввести команду

```
var_dump($document->getHeadData());
```

в метод `fetchHead()`, например, непосредственно перед кодом, инициирующим событие, а затем отобразить начальную страницу на веб-сайте, то на нее будет выведен длинный перечень результатов выполнения метода `getHeadData()`, частично приведенных ниже.

```
array
    'title' =>string 'Home' (length=4)
    'description' => string 'My description.' (length=15)
    'link' => string '' (length=0)
    'metaTags' =>
      array
        'http-equiv' =>
           array
             'content- type' => string 'text/html' (length=9)
        'standard' =>
           array
             'robots' => string 'index, follow' (length=13)
             'keywords' => string 'My keywords.' (length=12)
             'rights' => string 'My rights.' (length=10)
               'language' => string 'en-GB' (length=5)
```

Если сравнить этот фрагмент кода с приведенным ранее исходным кодом HTML, то можно обнаружить, что элементы метаданных с атрибутами `name` хранятся в объекте в виде ассоциативного массива, который, в свою очередь, хранится в элементе разметки `standard` элемента разметки `metaTags`. Значение атрибута `name` (например, `'robots'`) является ключом к ассоциативному массиву, а значение атрибута `content` (в частности, `'index, follow'`) — значением для ассоциативного массива.

Нам требуется, чтобы подключаемый модуль вводил новый элемент разметки метаданных с атрибутом `name`, имеющим значение `"revised"`, т.е. "исправлено", а также значение, которое может быть введено пользователем в качестве параметра в экранной форме компонента Plugin Manager. Кроме того, нам требуется сохранить любые существующие элементы разметки метаданных, добавив к ним лишь один новый элемент. Для этого подключаемый модуль должен выполнять следующее.

1. Считывать имеющиеся данные заголовка из документа. Эти данные могут быть представлены массивом, аналогичным приведенному ранее.
2. Добавить элемент в ассоциативный массив, хранящийся в элементе `standard` разметки массива внутри элемента разметки `metaTags`. Ключ этого массива будет "исправлен", а данные — введены пользователем в качестве параметра.
3. Записать видоизмененный массив обратно в объект документа, используя метод `setHeaderData()`, являющийся зеркальным отражением метода `getHeaderData()`, дополняя его.
4. И наконец, сделать все это только при наличии данных, введенных в качестве параметра подключаемого модуля, а иначе — ничего не делать.

Итак, перейдем непосредственно к разработке подключаемого модуля. Для этого выполните следующие действия.

1. Создайте папку для нового подключаемого модуля. Наш подключаемый модуль будет называться `mymeta`, поэтому создайте папку `plugins/system/mymeta`.
2. Для сохранения вводимых данных можно скопировать ряд уже имеющихся файлов и только поправить их. В частности, скопируйте файлы `index.html`, `p3p.php` и `p3p.xml` из папки `plugins/system/p3p` в новую папку `plugins/system/mymeta`. Затем переименуйте файлы `p3p.php` и `p3p.xml` на `mymeta.php` и `mymeta.xml` соответственно.
3. Внесите поправки в файл `mymeta.xml` так, как показано в листинге 5.2. В данном случае требуется внести изменения в XML-дескрипторы имени (`name`), автора (`author`), даты создания (`creationDate`), сведений об авторском праве (`copyright`), описания (`description`) и имени файла (`filename`).

Листинг 5.2. Исходный код из файла `mymeta.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<install version="1.6" type="plugin" group="system">
  <name>My Meta Plugin</name>
  <author>Mark Dexter and Louis Landry</author>
  <creationDate>January 2012</creationDate>
```

```
  <copyright>Copyright (C) 2012 Mark Dexter and Louis Landry.
↳All rights reserved.</copyright>
  <license>GNU General Public License version 2 or later; see LICENSE.txt
</license>
  <authorEmail>admin@joomla.org</authorEmail>
  <authorUrl>www.joomla.org</authorUrl>
  <version>2.5.0</version>
  <description>My Meta Plugin</description>
  <files>
    <filename plugin="mymeta">mymeta.php</filename>
    <filename>index.html</filename>
  </files>
  <config>
    <fields name="params">
      <fieldset name="basic">
        <field name="revised" type="text"
        description="Meta revised text for content attribute"
        label="Revised Content"
        default=""
        size="50"
        />
      </fieldset>
    </fields>
  </config>
</install>
```

Как показано выше, необходимо изменить полностью элемент разметки `field`, чтобы иметь возможность вводить новый параметр. Для этого установите значение `revised` в атрибуте `name`, а также значение `text` в атрибуте `type`. Кроме того, нужно установить соответствующие значения в атрибутах `description`, `label` и `size` данного элемента разметки.

4. Итак, код для организации ввода параметра подключаемого модуля уже имеется. Далее необходимо написать сам подключаемый модуль. В листинге 5.3 приведен исходный код подключаемого модуля, сохраняемый в файле `mymeta.php`.

Листинг 5.3. Исходный код из файла `mymeta.php`

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry.
 * @license       GNU General Public License version 2 or later; see LICENSE.
txt
 */
// прямой доступ отсутствует
defined('_JEXEC') or die;
jimport('joomla.plugin.plugin');
/**
 * Пример системного подключаемого модуля
 */
class plgSystemMyMeta extends JPlugin
{
    function onBeforeCompileHead()
    {
```

```
    if ($this->params->get('revised')) {
      $document = JFactory::getDocument();
      $headData = $document->getHeadData();
      $headData['metaTags']['standard']['revised'] =
        $this->params->get('revised');
      $document->setHeadData($headData); }
  }
}
```

В приведенном выше исходном коде класс переименован на `plugSystemMyMeta`, а метод назван `onBeforeCompileHead()` по имени события, обрабатываемого создаваемым здесь подключаемым модулем. Его исходный код нетрудно понять, если знать структуру массива, обрабатываемого в методе `getHeadData()`.

Сначала проверяется наличие введенных данных в "исправленном" параметре подключаемого модуля. Если они отсутствуют, то вся обработка пропускается, а если они присутствуют, то она выполняется. Для этого получается объект документа `$headData`, где и сохраняются результаты обработки в методе `getHeadData()`.

Далее создается новый элемент "исправлено" ассоциативного массива и в нем устанавливается введенное значение параметра подключаемого модуля. Напомним, что этот массив вложен в два других массива, как было показано ранее при выводе результатов выполнения метода `getHeadData()`.

5. Итак, подключаемый модуль завершен и готов к применению. Но он пока еще неизвестен в установке Joomla. И хотя файлы находятся в нужных папках, для нового подключаемого модуля отсутствует строка в таблице `#__extensions`.

   Напомним, что при копировании шаблона `beez20` в главе 4 мы воспользовались средством Discover для установки нового шаблона. То же самое придется сделать и теперь. Итак, войдите в административную часть веб-сайта, перейдите по команде Extensions⇨Extension Manager⇨Discover и выберите вкладку Discover. Затем щелкните на пиктограмме Discover непосредственно на панели инструментов, чтобы получить вид, аналогичный приведенному на рис. 5.1.

6. Щелкните сначала на флажке слева от имени нового подключаемого модуля, а затем на пиктограмме Install непосредственно на панели инструментов. В итоге должно появиться сообщение об успешной установке подключаемого модуля.

   При установке нового подключаемого модуля в качестве расширения в таблице `#__extensions` создается строка, в которой сохраняются сведения о данном подключаемом модуле. В Joomla известны только те расширения, которые перечислены в этой таблице. А в результате действия средства Discover обнаруживаются те расширения, которые находятся в файловой системе, а не в таблице `#__extensions`. Как правило, расширение устанавливается из архивного файла, специально созданного для этой цели. В следующем разделе будет создан подключаемый модуль и упакованный архив для его установки.

А теперь, когда подключаемый модуль установлен, проверим его в действии. С этой целью перейдите по команде Extensions⇨Plugin Manager и отфильтруйте системные подключаемые модули. Среди них должен появиться вновь созданный нами подключаемый модуль `My Meta Plugin`. Щелкните на нем, чтобы перейти к экрану, приведенному на рис. 5.2.

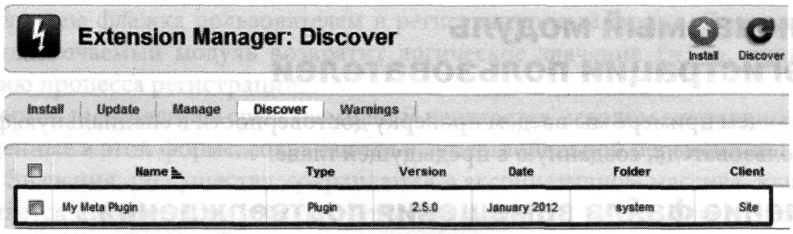


Рис. 5.1. Вид экрана обнаружения расширений Joomla с новым подключаемым модулем

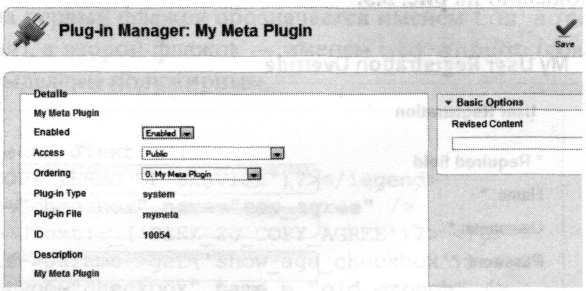


Рис. 5.2. Экран редактирования подключаемого модуля `My Meta Plugin`

Активизируйте подключаемый модуль, выбрав вариант Enabled из одноименного списка, введите любые данные в поле Revised Content (Исправленное содержимое), а затем перейдите на начальную страницу веб-сайта. Выберите в своем браузере режим отображения страницы в исходном коде HTML (например, по команде меню Инструменты⇨Веб-разработка⇨Исходный код страницы в браузере Firefox). В окне браузера должен появиться исходный код, аналогичный приведенному ниже. Добавленная в нем строка выделена полужирным.

```
<meta name="robots" content="index, follow" />
<meta name="keywords" content="My keywords." />
<meta name="rights" content="My rights." />
<meta name="language" content="en- GB" />
<meta name="revised" content="Mark Dexter, 17 March 2011" />
<meta name="description" content="My description." />
```

В качестве завершающей проверки перейдите к редактору подключаемых модулей (Plugin Editor) и очистите поле Revised Content. Затем отобразите еще раз начальную страницу веб-сайта и проверьте ее исходный код, как пояснялось выше. Теперь дескриптор метаданных с атрибутом `name="revised"` должен отсутствовать, поскольку отсутствует содержимое, помеченное этим дескриптором.

По зрелом размышлении можно по достоинству оценить, насколько просто было внести описанные выше изменения с помощью подключаемого модуля. Для этого оказалось достаточно добавить в систему два новых файла и отредактировать несколько строк кода. Учитывая столь малые затраты труда, можно было бы аналогичным образом изменить содержимое элемента разметки заголовка на каждой странице веб-сайта.

Подключаемый модуль для регистрации пользователей

В следующем примере мы введем проверку достоверности в специальную форму регистрации пользователя, созданную в предыдущей главе.

Обновление файла замещения подтверждения

Напомним, что ранее мы добавили два флажка в специальную форму регистрации пользователя, как показано на рис. 5.3.

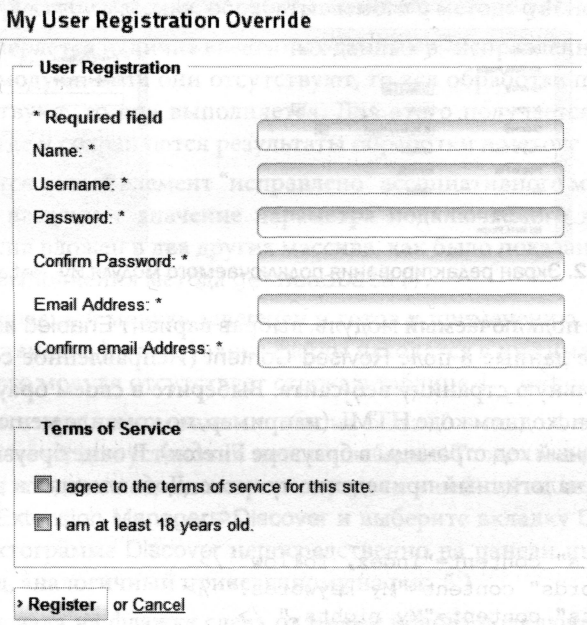


Рис. 5.3. Специальная регистрационная форма

Для этого в файл замещения компоновки `templates/beez_20_copy/html/com_users/registration/approval.php` был введен следующий фрагмент кода:

```
<fieldset>
    <legend><?php echo JText::_('BEEZ_20_COPY_TERMS_OF_SERVICE')?></legend>
    <p><input type="checkbox" />
      <?php echo JText::_()?> </p>
    <?php if ($this- >params- >get('show_age_checkbox')) : ?>
      <p><input type="checkbox" />
        <?php echo JText::_('BEEZ_20_COPY_AGE')?> </p>
    <?php endif; ?>
</fieldset>
```

Этот фрагмент кода придется немного видоизменить, прежде чем приступать к написанию подключаемого модуля, в котором должно быть проверено, установлены ли оба

упомянутых выше флажка пользователем в регистрационной форме. Если они не установлены, подключаемый модуль возвратит логическое значение false, приводящее к прекращению процесса регистрации.

Когда форма, составленная средствами PHP, передается методом отправки (POST), значения, введенные в этой форме, сохраняются в суперглобальной переменной $_REQUEST языка PHP. Значения, по существу, сохраняются в ассоциативном массиве, ключом к которому является атрибут name каждого элемента разметки ввода. Если у элемента разметки ввода отсутствует атрибут name, его значение не сохраняется. Поэтому нам нужно дополнить поля обоих рассматриваемых здесь флажков атрибутами name. В приведенном ниже фрагменте кода первый флажок обозначается именем tos_agree (согласие с условиями обслуживания), а второй флажок — именем old_enough (совершеннолетие), где оба атрибута name выделены полужирным.

```
<fieldset>
    <legend><?php echo JText::_(
        'BEEZ_20_COPY_TERMS_OF_SERVICE')?></legend>
    <p><input type="checkbox" name="tos_agree" />
        <?php echo JText::_('BEEZ_20_COPY_AGREE')?> </p>
    <?php if ($this->params->get('show_age_checkbox')) : ?>
        <p><input type="checkbox" name = "old_enough" />
            <?php echo JText::_('BEEZ_20_COPY_AGE')?> </p>
    <?php endif; ?>
</fieldset>
```

Ввод исходного XML-файла

Перейдем к созданию исходных PHP- и XML-файлов для подключаемого модуля. Этот модуль носит название myregistration и относится к типу пользовательских подключаемых модулей. Поэтому создадим для него папку plugins/user/myregistration, а в ней — два исходных файла myregistration.xml и myregistration.php. В листинге 5.4 приведен исходный код из файла myregistration.xml.

Листинг 5.4. Исходный код из файла myregistration.xml

```
<?xml version="1.0" encoding="utf-8"?>
<extension version="2.5" type="plugin" group="user">
    <name>plg_user_myregistration</name>
    <author>Mark Dexter and Louis Landry</author>
    <creationDate>January 2012</creationDate>
    <copyright>(C) 2012 Mark Dexter and Louis Landry.</copyright>
    <license>GNU General Public License version 2 or later; see
↳LICENSE.txt</license>
  <authorEmail>admin@joomla.org</authorEmail>
  <authorUrl>www.joomla.org</authorUrl>
  <version>2.5.0</version>
  <description>PLG_USER_MYREGISTRATION_XML_DESCRIPTION</description>

  <files>
    <filename plugin="myregistration">myregistration.php</filename>
    <filename>index.html</filename>
    <folder>language</folder>
```

```
  </files>

  <config>
  </config>
</extension>
```

Этот код подобен исходному коду XML из предыдущего примера подключаемого модуля. Обратите внимание на определение подпапки языковой поддержки в данном подключаемой модуле. Мы еще вернемся к этому вопросу, когда дело дойдет до создания файла упакованного архива.

Ввод исходного PHP-файла

В листинге 5.5 приведен исходный код из файла `myregistration.php`.

Листинг 5.5. Исходный код из файла `myregistration.php`

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter & Louis Landry. All
↪rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.
txt
 */

defined('JPATH_BASE') or die;
jimport('joomla.plugin.plugin');

/**
 * Это специальный класс подключаемого модуля регистрации.
 * В нем проверяется, установил ли пользователь флажки, указывающие
 * на то, что он согласен с условиями обслуживания и достаточно взрослый,
 * чтобы иметь право пользоваться этим сайтом.
 */
class plgUserMyRegistration extends JPlugin
{
  /**
   * Метод для обработки события onUserBeforeSave и определения
   * достоверности введенных данных для их последующего сохранения.
   * В нем, в частности, проверяется, сохраняются ли регистрационные
   * данные нового пользователя и установлены ли им флажки, указывающие
   * на то, что он согласен с условиями обслуживания и достаточно
   * взрослый, чтобы иметь право пользоваться этим сайтом.
   *
   * @param array $previousData     Сохраняемые в настоящий момент
   *                                регистрационные данные пользователя.
   * @param boolean     $isNew      Истинно, если сохраняются регистрационные
   *                                данные нового пользователя.
   * @param array $futureData       Вновь сохраняемые регистрационные
   *                                данные пользователя.
   *
   * @return      boolean           Истинно, если требуется продолжить
   *                                процесс сохранения. Ложно, если нужно
   *                                прекратить этот процесс.
   *
```

```
     * @since 1.0
    */

    function onUserBeforeSave($previousData, $isNew, $futureData)
    {
      // Если регистрационные данные нового пользователя не сохраняются
      // или если это не пользовательская часть веб-сайта, выполнить процесс
      // сохранения без прерывания.
      if (!$isNew || !JFactory::getApplication()->isSite()) {
        return true;
      }

      // загрузить файл языковой поддержки подключаемого модуля
      $this->loadLanguage();
      $result = true;

      // проверить, установлен ли флажок, указывающий на то,
      // что пользователь согласен с условиями обслуживания
      if (!JRequest::getBool('tos_agree')) {
          JError::raiseWarning(1000,
              JText::_('PLG_USER_MYREGISTRATION_TOS_AGREE_REQUIRED'));
          $result = false;
      }

      // проверить, установлен ли флажок, указывающий на то, что
      // пользователю уже, по крайней мере, 18 лет
      if (!JRequest::getBool('old_enough')) {
          JError::raiseWarning(1000,
              JText::_('PLG_USER_MYREGISTRATION_OLD_ENOUGH_REQUIRED'));
          $result = false;
      }

      return $result;
    }
}
```

Две первые строки кода должны быть вам уже знакомы. Сначала нужно убедиться в том, что мы находимся в среде Joomla, а затем импортировать родительский класс для данного подключаемого модуля. (Следует, однако, иметь в виду, что вторая рассматриваемая здесь строка кода уже не нужна, поскольку в версии Joomla 2.5 для этой же цели используется автозагрузчик.) Имя класса указывается в соответствии с принятыми условными обозначениями: `plg` (подключаемый модуль), `User` (пользовательский тип подключаемого модуля), `MyRegistration` (имя подключаемого модуля). Этот класс расширяет класс `JPlugin`.

У класса подключаемого модуля имеется один метод, обозначаемый по имени обрабатываемого события, а в данном случае — `onUserBeforeSave()`. Это событие инициируется при попытке сохранить регистрационные данные нового пользователя в административной части веб-сайта или зарегистрировать его в пользовательской части. Прежде всего необходимо убедиться в том, что в пользовательской части веб-сайта создается новый пользователь. В противном случае возвращается логическое значение `true` и остальная часть обработки пропускается.

Затем загружается файл языковой поддержки. В данном случае это файл `administrator/language/en-GB/en-GB.plg_user_myregistration.ini`, о котором речь пойдет несколько ниже. А после этого в переменной `$result` устанавливается логическое значение `true`.

Далее следует кодовый блок условного оператора `if`. В этом блоке метод `JRequest::getBool()` вызывается для получения элемента ввода `tos_agree` из суперглобальной переменной `$_REQUEST` языка PHP. Этот метод возвращает логическое значение `true` или `false`, поскольку данный элемент ввода является флажком и может находиться в установленном или сброшенном состоянии. Но мы должны принять во внимание и то обстоятельство, что злоумышленник может тем или иным способом манипулировать переменной `$_REQUEST`, присваивая ей совершенно неожиданные значения. Поэтому, вызывая метод `JRequest::getBool()`, мы твердо знаем, что он возвратит логическое значение `true` или `false`, независимо от того, что именно злоумышленник может ввести в поле данного элемента ввода.

Если флажок установлен, то после вызова метода `JRequest::getBool(tos_agree)` возвратится логическое значение `true`, тогда как выражение `!JRequest::getBool('tos_agree')` окажется ложным (`false`) вследствие применения оператора отрицания (`!`). В данном случае код внутри блока не выполняется.

Если же флажок не установлен, то выполняются две строки кода в блоке условного оператора `if`. В первой из них вызывается метод `JError::raiseWarning()`. В качестве его первого аргумента указывается код ошибки, хотя в данном случае он не используется и поэтому может быть каким угодно, а в качестве второго аргумента — текст сообщения об ошибке. Для перевода этого текста в данном случае используется метод `JText::_()`. А это означает, что нам придется ввести языковый ключ `PLG_USER_MYREGISTRATION_TOS_AGREE_REQUIRED` в файл языковой поддержки. Во второй строке кода в переменной `$result` устанавливается логическое значение `false`. И это означает, что рассматриваемый здесь метод возвратит логическое значение `false`, чтобы прекратить процесс сохранения регистрационных данных пользователя.

Второй условный оператор `if` похож на первый, за исключением того, что в нем проверяется состояние второго флажка и пользователю возвращается другое сообщение. А в последней строке кода просто возвращается значение переменной `$result`, которое будет истинным, если пропущены кодовые блоки обоих условных операторов `if`, в которых проверяется состояние упомянутых выше флажков. Так, если пользователь забудет установить эти флажки, он получит соответствующие сообщения об ошибках, что нам, собственно, и требуется.

Ввод файлов языковой поддержки

На последней стадии перед опробованием рассматриваемого здесь метода вводятся файлы языковой поддержки. Напомним, что в исходный XML-файл были введены следующие строки кода:

```
<files>
  <filename plugin="myregistration">myregistration.php</filename>
  <filename>index.html</filename>
  <folder>language</folder>
</files>
```

В элементе разметки `folder` указывается на наличие подпапки `language` в папке рассматриваемого здесь подключаемого модуля. При его создании можно выбрать наличие файлов языковой поддержки в папке подключаемых модулей или же в папке `adminstrator/languages`. Файлы всех расширений обычно рекомендуется хранить отдельно от основных файлов, и поэтому файлы языковой поддержки зачастую предпочтительно размещать в папке с расширениями.

В рассматриваемом здесь примере имеются два файла языковой поддержки: `en-GB.plg_user_myregistration.ini` и `en-GB.plg_user_myregistration.sys.ini`. Оба эти файла должны быть размещены в папке `plugins/user/myregistration/language/en-GB/`.

Первый из них является основным файлом языковой поддержки и содержит языковые ключи, применяемые при выполнении кода подключаемого модуля, а также при его открытии для редактирования в компоненте Plugin Manager. В этот файл вводятся любые ключи, которые могут потребоваться для отображения данных в пользовательской части веб-сайта или для редактирования параметров. В листинге 5.6 приведено содержимое основного файла языковой поддержки.

Листинг 5.6. Содержимое файла `en-GB.plg_user_myregistration.ini`

```
; Файл языковой поддержки подключаемого модуля myregistration plugin

PLG_USER_MYREGISTRATION_TOS_AGREE_REQUIRED="You must agree to the
↳terms of service."
PLG_USER_MYREGISTRATION_OLD_ENOUGH_REQUIRED="You must be at least
↳18 years old."
```

Второй системный файл языковой поддержки (с расширением имени `.sys`) служит для перевода названия подключаемого модуля, когда он перечисляется в компоненте Extension Manager или Plugin Manager. В этот файл вводится также описание подключаемого модуля для его перевода в сообщении, которое выводится после установки подключаемого модуля. Это правило распространяется на все типы расширений. В листинге 5.7 приведено содержимое системного файла языковой поддержки.

Листинг 5.7. Содержимое файла `en-GB.plg_user_myregistration.sys.ini`

```
; Системный файл языковой поддержки подключаемого модуля myregistration
; Системные файлы с расширением .sys.ini используются при составлении списков
; расширений в диспетчере расширений или диспетчере подключаемых модулей

PLG_USER_MYREGISTRATION="User - My Registration"
PLG_USER_MYREGISTRATION_XML_DESCRIPTION="Checks that terms and age boxes
↳have been checked."
```



И в завершение скопируйте файл `index.html` из другой папки Joomla в папку `plugins/user/myregistration`, а также в подпапки `language` и `language/en-GB`. Как пояснялось ранее, каждая папка, создаваемая в Joomla, должна содержать файл `index.html`, чтобы пользователи не могли просматривать папки непосредственно.

Проверка подключаемого модуля

А теперь проверим созданный нами подключаемый модуль. С этой целью снова перейдите по команде Extensions⇨Extension Manager⇨Discover и щелкните на пиктограмме Discover непосредственно на панели инструментов. Вновь созданный подключаемый модуль должен быть перечислен как расширение с помощью текста "User - My Registration" (Пользовательский — Моя регистрация), переведенного на британский английский в системном файле языковой поддержки с расширением .sys.

Совет: удаление подключаемого модуля средствами phpMyAdmin

При тестировании или отладке подключаемого модуля или другого расширения может возникнуть потребность повторить стадии обнаружения и установки. В таком случае, возможно, придется сначала удалить расширение, чтобы тем самым удалить файлы этого расширения из папок Joomla.

Для этого достаточно удалить строку с данным расширением из таблицы #__extensions в базе данных, например, средствами phpMyAdmin. Благодаря этому установка расширения отменяется без удаления его файлов. Если вы не знакомы с базой данных MySQL и программой phpMyAdmin, то подробнее они рассматриваются в главе 11.

Щелкните сначала на флажке слева от названия подключаемого модуля, а затем на пиктограмме Install. После установки подключаемого модуля его описание из файла системного языковой поддержки должно появиться на экране Extension Manager: Discover, как показано на рис. 5.4.

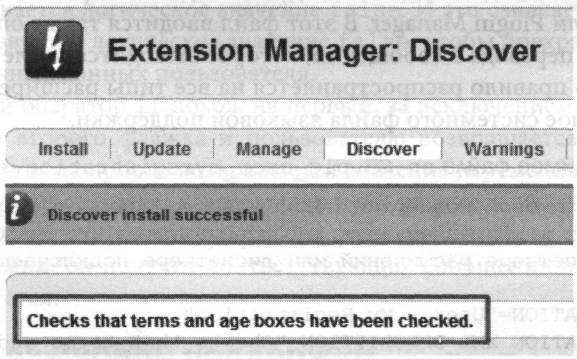


Рис. 5.4. Описание подключаемого модуля из системного файла языковой поддержки

Для того чтобы проверить подключаемый модуль, сначала активизируйте его в компоненте Plugin Manager, а затем попробуйте зарегистрировать нового пользователя в регистрационной форме из файла замещения approval.php, но не устанавливая в ней оба флажка. В итоге должно появиться сообщение, приведенное на рис. 5.5.

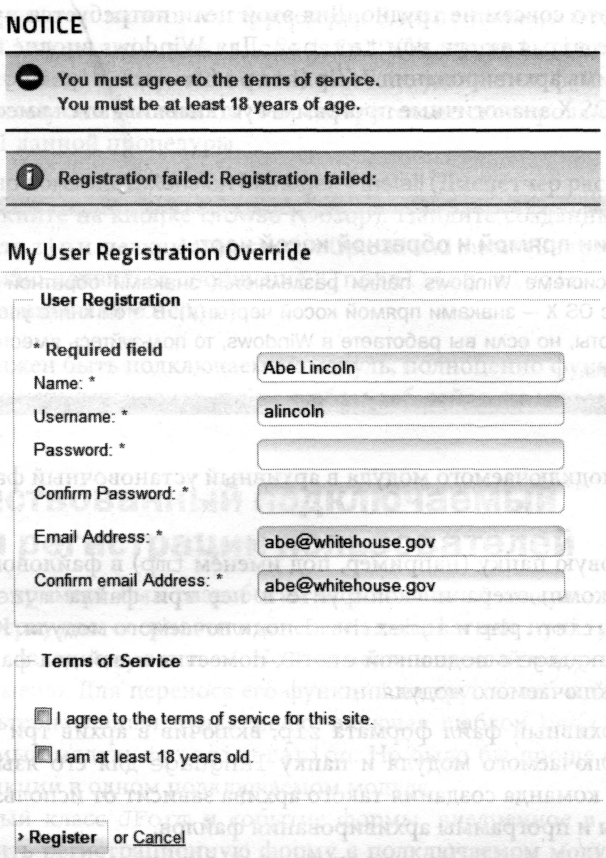


Рис. 5.5. Сообщения об ошибках, выдаваемые пользовательским подключаемым модулем

Работоспособность подключаемого модуля следует проверить и на других контрольных примерах, включая следующие.

- Сохранение регистрационных данных с одним установленным флажком (в этом случае должно быть выдано одно сообщение об ошибке).
- Сохранение регистрационных данных с обоими установленными флажками (в этом случае подключаемый модуль должен работать корректно).
- Создание нового пользователя в административной части веб-сайта (в этом случае команда Users⇨User Manager⇨Add⇨New User (Пользователи⇨Диспетчер пользователей⇨Добавить нового пользователя) должна быть выполнена корректно).

Упаковка подключаемого модуля

До сих пор мы пользовались средством Discover для установки расширений, что вполне подходит для стадии разработки. Но если нам потребуется установить расширение на других веб-сайтах Joomla, то придется упаковать его в архивный установочный файл.

Впрочем, сделать это совсем не трудно. Для этой цели потребуется программа создания архивов в формате `zip`, `tar.gz` или `tar.bz2`. Для Windows вполне подойдет свободно доступная программа архивирования 7-Zip (`http://www.7-zip.org/download.html`), а для Linux и Mac OS X аналогичные программы устанавливаются вместе с операционной системой.

Об употреблении прямой и обратной косой черты

В файловой системе Windows папки разделяются знаками обратной косой черты (\), а в Linux или Mac OS X — знаками прямой косой черты (/). В этой книге употребляются знаки прямой косой черты, но если вы работаете в Windows, то пользуйтесь вместо них знаками обратной косой черты.

Для упаковки подключаемого модуля в архивный установочный файл выполните следующие действия.

1. Создайте новую папку (например, под именем `tmp`) в файловой системе на жестком диске компьютера и скопируйте в нее три файла `myregistration.xml`, `myregistration.php` и `index.html` подключаемого модуля. Кроме того, создайте папку `language` с подпапкой `en-GB`, поместив в ней два файла языковой поддержки подключаемого модуля.
2. Создайте архивный файл формата `zip`, включив в архив три упомянутых выше файла подключаемого модуля и папку `language` для его языковой поддержки. Конкретная команда создания такого архива зависит от используемой операционной системы и программы архивирования файлов.

   Для архивирования средствами 7-Zip под Windows выделите три файла подключаемого модуля и папку его языковой поддержки в Проводнике Windows, щелкните правой кнопкой мыши, выберите команду 7-Zip⇨Добавить в архив и далее следуйте инструкциям, присвоив новому архиву имя, например, `plg_user_myregistration.zip`.

   Архивирование в Mac OS X выполняется аналогичным образом, за исключением того, что после выделения файлов из контекстного меню выбирается команда Create Archive (Создать архив).

   А в Linux достаточно выбрать папку `tmp`, перейти в режим командной строки и ввести следующую команду:

   ```
   $ zip -r plg_user_myregistration.zip *
   ```

3. Создав архив в формате `zip`, откройте его и проверьте наличие в нем трех файлов подключаемого модуля и папки его языковой поддержки.
4. А теперь нужно проверить правильность установки файлов подключаемого модуля. Удалите подключаемый модуль, перейдя по команде Extensions⇨Extension Manager⇨Manage (Расширения⇨Диспетчер расширений⇨Управлять). Выберите подключаемый модуль и щелкните на пиктограмме Uninstall (Удалить)

непосредственно на панели инструментов. В итоге должно появиться сообщение "Uninstalling plugin was successful" (Удаление выполнено успешно). Следует, однако, иметь в виду, что на данном этапе удаляются файлы из папок Joomla. Именно поэтому они должны быть предварительно скопированы в папку tmp, созданную в п.1 данной процедуры.

5. Перейдите по команде Extension Manager⇨Install (Диспетчер расширений⇨Установить) и щелкните на кнопке Browse (Обзор). Найдите созданный ранее архивный файл формата zip и щелкните на кнопке Upload and Install (Выгрузить и установить). В итоге должно появиться сообщение "Installing plugin was successful" (Установка выполнена успешно).

Теперь у вас должен быть подключаемый модуль, полноценно функционирующий как расширение. Вы можете установить его на любом веб-сайте, построенном на Joomla, начиная с версии 1.6.

Усовершенствованный подключаемый модуль для регистрации пользователей

В предыдущем примере был разработан подключаемый модуль myregistration с целью ввести проверку достоверности в качестве альтернативы пункту из меню регистрации пользователя, созданному в главе 4. Но этот подключаемый модуль зависит от альтернативного пункта меню. Для переноса его функций на другой веб-сайт Joomla пришлось бы установить альтернативный пункт меню, включая шаблон beez_20_copy, а также новый подключаемый модуль myregistration. Но было бы проще совместить все эти элементы и их функции в одном подключаемом модуле.

Используя новый класс JForm и событие формы, внедренное в версии Joomla 1.6, мы можем заместить регистрационную форму в подключаемом модуле, не создавая отдельно файл для альтернативного пункта меню. Кроме того, мы можем воспользоваться классом JForm для проверки достоверности, исключив тем самым потребность в методе onBeforeSave() из подключаемого модуля. При таком подходе нам удастся упаковать все эти функциональные возможности в одном небольшом подключаемом модуле в качестве расширения и значительно упростить их реализацию на других веб-сайтах Joomla.

Присвоим новой версии подключаемого модуля имя myregistration2. Он должен состоять из следующих файлов.

- forms/form.xml: файл со сведениями о классе JForm для полей, вводимых в подключаемом модуле.
- language/en-GB/en-GB.plg_user_myregistration2.ini: основной файл языковой поддержки.
- language/en-GB/en-GB.plg_user_myregistration2.sys.ini: системный файл языковой поддержки.
- myregistration2.php: исходный файл PHP подключаемого модуля.
- myregistration2.xml: XML-файл подключаемого модуля.

Итак, рассмотрим все стадии процесса создания нового подключаемого модуля.

Создание главного XML-файла для подключаемого модуля

Как и прежде, создайте папку для подключаемого модуля (`plugins/user/myregistration2`), а в ней — главный XML-файл. Его содержимое приведено в листинге 5.8.

Листинг 5.8. Исходный код из файла `myregistration2.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<extension version="2.5" type="plugin" group="user" method="upgrade" >
    <name>plg_user_myregistration2</name>
    <author>Mark Dexter and Louis Landry</author>
    <creationDate>January 2012</creationDate>
    <copyright>(C) 2012 Mark Dexter and Louis Landry. All rights reserved.
    </copyright>
    <license>GNU General Public License version 2 or later; see LICENSE.txt
    </license>
    <authorEmail>admin@joomla.org</authorEmail>
    <authorUrl>www.joomla.org</authorUrl>
    <version>2.5.0</version>
    <description>PLG_USER_MYREGISTRATION2_XML_DESCRIPTION</description>

    <files>
        <filename plugin="myregistration2">myregistration2.php</filename>
        <filename>index.html</filename>
        <folder>forms</folder>
        <folder>language</folder>
    </files>

    <config>
    </config>
</extension>
```

Этот файл подобен XML-файлу из предыдущего примера. Обратите внимание на то, что в нем изменено название подключаемого модуля, указываемое один раз в элементе разметки `name` и дважды в элементе разметки `filename`, а также добавлен элемент разметки `folder` для указания папки `forms`. Подробнее об этом речь пойдет в следующем разделе.

Создание XML-файла для разметки формы

В рассматриваемом здесь примере класс `JForm` используется для добавления двух полей в регистрационную форму. Это можно сделать одним из следующих способов:

- загрузить поля из XML-файла;
- загрузить поля из символьной строки, созданной, например, в исходном файле PHP подключаемого модуля.

Первый способ рекомендуется в большинстве случаев, поскольку он упрощает обращение с XML-файлом и его сопровождение. В листинге 5.9 приведен исходный код из XML-файла для разметки регистрационной формы.

Листинг 5.9. Исходный код из файла `form.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<form>
    <fieldset name="tos"
        label="PLG_USER_MYREGISTRATION2_TERMS_OF_SERVICE"
    >
        <field name="tos_agree" type="checkbox"
            default="0"
            filter="bool"
            label="PLG_USER_MYREGISTRATION2_AGREE"
            required="true"
            value="1"
        />

        <field name="old_enough" type="checkbox"
            default="0"
            filter="bool"
            label="PLG_USER_MYREGISTRATION2_AGE"
            required="true"
            value="1"
        />
    </fieldset>
</form>
```

В этом XML-файле определены два поля, которые нам требуется ввести в форму регистрации пользователя. Соответствующий исходный код очень похож на код HTML, формируемый средствами класса `JForm`. Внешний элемент разметки в этом файле называется `form` и служит для создания элемента `form` разметки в коде HTML. Он содержит элемент `fieldset` разметки в коде HTML, который служит для группирования полей в форме. А внутри этого элемента находятся еще два элемента разметки `field`, определяющие нужные нам поля для регистрационной формы.

У каждого из этих полей имеются следующие атрибуты.

- `default`: устанавливаемое по умолчанию значение, если флажок сброшен.
- `filter`: фильтр, используемый для проверки данных, введенных в этом поле.
- `label`: метка поля (как правило, переводится).
- `required`: признак, указывающий классу `JForm` сделать это поле обязательным для заполнения.
- `value`: значение, задаваемое в форме при установке флажка.

Атрибуты `label` и `value` являются стандартными для элемента разметки полей ввода в коде HTML. Атрибут `filter` вынуждает класс `JForm` отфильтровать поле ввода данных, используя одно из стандартных значений из класса `JHtml`. В данном случае должны отфильтровываться только логические значения `true` и `false`. И даже если пользователь изменит форму в своем браузере, чтобы передать другую информацию (например, злоумышленный код запроса SQL или сценария JavaScript), эта информация будет отфильтрована средствами класса `JForm` и преобразована в соответствующее логическое значение.

Атрибут `default` определяет значение, передаваемое при условии, что данные не введены (в данном случае не установлен флажок). В качестве значения по умолчанию указан нуль, который преобразуется в логическое значение `false`.

Атрибут `required` указывает классу `JForm` на то, что поле должно быть заполнено обязательно (в данном случае установлен флажок). Класс `JForm` не даст пользователю возможность зарегистрироваться, если он не установит флажок. Но поскольку эта проверка достоверности выполняется в классе `JForm` автоматически, то нам не потребуется метод `onBeforeSave()` в подключаемом модуле `myregistration`.

В следующем разделе будет показано, как используется этот XML-файл.

Создание исходного файла PHP для подключаемого модуля

В листинге 5.10 приведен исходный код из файла PHP для рассматриваемого здесь подключаемого модуля.

Листинг 5.10. Исходный код из файла `myregistration2.php`

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

defined('JPATH_BASE') or die;
/**
 * Это класс специального подключаемого модуля регистрации.
 * В нем проверяется, установил ли пользователь флажки, указывающие
 * на то, что он согласен с условиями обслуживания и достаточно
 * взрослый, чтобы пользоваться этим сайтом.
 *
 * @package       Joomla.Plugins
 * @subpackage    User.MyRegistration2
 * @since         1.0
 */
class plgUserMyRegistration2 extends JPlugin
{
    /**
     * Метод для обработки события onContentPrepareForm и внесения
     * изменений в форму регистрации пользователя.
     * В нем сначала проверяется, является ли подготавливаемая форма той,
     * что выбирается из компонента com_users для регистрации пользователя.
     * Если это именно такая форма, специальный xml-файл загружается в
     * объект формы для добавления в нее соответствующих полей.
     * @param     JForm $form       Изменяемая форма.
     * @param     array $data       Данные, связанные с формой.
     *
     * @return    boolean
     *
     * @since     1.0
    */
  public function onContentPrepareForm($form, $data)
  {
    // если это не регистрационная форма, проигнорировать ее
```

```
    if ($form->getName() != 'com_users.registration') {
       return;
    }

    // загрузить файл языковой поддержки подключаемого модуля
    $this->loadLanguage();

    // загрузить специальный xml-файл в форму регистрации пользователя
    $form->loadFile(dirname(__FILE__).'/forms/form.xml');
  }
}
```

Первая часть приведенного выше исходного кода ничем особенным не отличается от исходного кода рассмотренных ранее подключаемых модулей. Импорт класса `JPlugin` опирается на автозагрузчик, а далее этот класс расширяется. Получающемуся в итоге классу подключаемого модуля присваивается имя `plgUserMyRegistration2` в соответствии с принятыми условными обозначениями.

У данного класса имеется единственный метод `onContentPrepareForm()`. Событие `onContentPrepareForm` инициируется в тот момент, когда форма из объекта типа `JForm` подготовлена, но еще не воспроизведена. Мы можем видоизменить объект типа `JForm` в рабочей области оперативной памяти, прежде чем воспользоваться им для создания формы. Упомянутому выше методу передаются два аргумента: переменная `$form`, в которой хранится объект типа `JForm`, а также переменная `$data`, в которой хранится стандартный объект с любыми данными для заполнения формы.

Затем проверяется, является ли обрабатываемая форма регистрационной. Если она таковой не является, а следовательно, обрабатывается другая форма, происходит возврат из данного метода. С этой целью форма проверяется по ее названию. Если оно не совпадает с именем `com_users.registration`, происходит возврат из метода без всякой обработки данных.

Итак, мы выяснили, что обрабатываемая форма является регистрационной. Поэтому далее загружается файл языковой поддержки для перевода текста надписей в форме на избранный язык. А вся работа по созданию двух новых полей выполняется в самой последней строке кода. С этой целью вызывается метод `loadFile()` из класса `JForm` с файлом `form.xml` в качестве аргумента. В итоге поля из файла `form.xml` добавляются в форму, которая уже загружена в рабочую область оперативной памяти из стандартного XML-файла (в данном случае — `components/com_users/models/forms/registration.xml`). А поскольку оба поля из файла `form.xml` оказываются новыми для данной формы, т.е. они отличаются своими именами от других полей формы, то они беспрепятственно добавляются в эту форму.

Вот, собственно, и все. Итак, новые поля из файла `form.xml` добавлены в форму регистрации пользователя, а следовательно, они включаются в общий вывод. Как упоминалось ранее, дополнительный код для проверки факта установки обоих флажков в данной форме не требуется, поскольку в их полях используется атрибут `required`, предписывающий выполнить такую проверку автоматически средствами класса `JForm`.

Ввод файлов языковой поддержки

Как и прежде, у нас имеются два файла языковой поддержки, находящиеся в папке `plugins/user/myregistration2/language/en-GB`). Основным является файл `en-GB.plg_user_myregistration2.ini`, тогда как системный файл (с расширением `.sys`)

служит для перевода названия и описания подключаемого модуля в диспетчере подключаемых модулей. Содержимое этих файлов приведено в листингах 5.11 и 5.12 соответственно.

Листинг 5.11. Содержимое файла `n-GB.plg_user_myregistration2.ini`

```
; Основной файл языковой поддержки подключаемого модуля myregistration2
PLG_USER_MYREGISTRATION2_TERMS_OF_SERVICE="Added Fields for Terms
↳of Service Agreement"
PLG_USER_MYREGISTRATION2_AGREE="I agree to the terms."
PLG_USER_MYREGISTRATION2_AGE="I am at least 18 years old."
```

Листинг 5.12. Содержимое файла `en-GB.plg_user_myregistration2.sys.ini`

```
; Системный файл языковой поддержки подключаемого модуля myregistration2
; Системные файлы с расширением .sys.ini используются при составлении списков
; расширений в диспетчере расширений или диспетчере подключаемых модулей
PLG_USER_MYREGISTRATION2="User - My Registration2"
PLG_USER_MYREGISTRATION2_XML_DESCRIPTION="Demonstration plugin that checks
↳that overrides user registration. Checks that terms and age boxes have
↳been checked."
```

Проверка подключаемого модуля

Проверьте вновь созданный подключаемый модуль таким же образом, как и предыдущие, используя функции Discover и Install. С этой целью выключите из работы подключаемый модуль `myregistration`, активизируйте подключаемый модуль `myregistration2` и убедитесь в том, что в меню `Registration` используется исходный тип пункта меню, а не альтернативный тип (`Registration with Approval`), созданный в главе 4. Приняв эти подготовительные меры, вы увидите на экране регистрационную форму после загрузки пункта меню `Registration`, как показано на рис. 5.6.

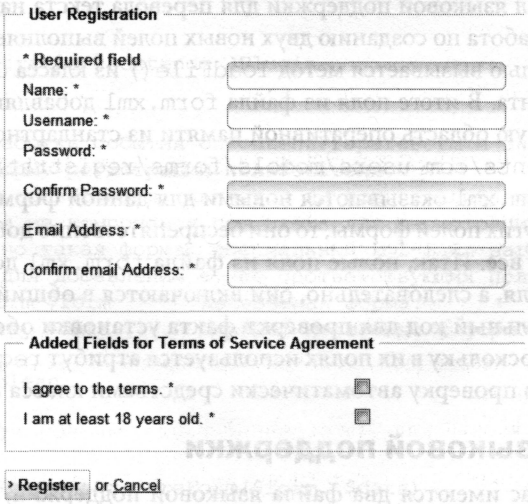


Рис. 5.6. Специальная регистрационная форма из подключаемого модуля `myregistration2`

Обратите внимание на то, что оба новых поля обозначены знаками звездочки, а это означает, что они, как и остальные поля регистрационной формы, обязательны для заполнения. Так, если вы щелкнете на кнопке `Register`, не установив оба флажка в этих полях, на экране появятся сообщения об ошибках, приведенные на рис. 5.7. Это стандартные сообщения, выводимые средствами класса `JForm`, когда пользователь передает форму, пропустив в ней обязательные для заполнения поля.

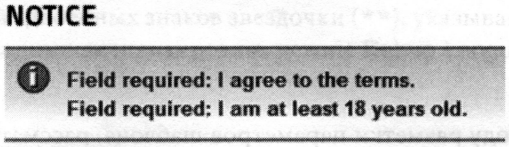


Рис. 5.7. Сообщения, появляющиеся в том случае, если не установлены флажки в обязательных для заполнения полях регистрационной формы

Упаковка подключаемого модуля

Для того чтобы упаковать подключаемый модуль с целью его последующей установки из архивного файла, выполните описанные ранее действия. Ниже дается краткое их описание.

1. Скопируйте файлы подключаемого модуля из папки `plugins/user/myregistration2` во временную папку на жестком диске компьютера.
2. Воспользуйтесь программой архивирования для создания из этих файлов архива, упакованного в формате `zip`. В соответствии с принятыми условными обозначениями файлу этого архива следует присвоить имя `plg_user_myregistration2.zip`, хотя вы вольны назвать его как угодно — главное, чтобы он содержал нужные файлы.

Удалите работающий экземпляр подключаемого модуля `yregistration2` и проверьте, насколько успешно он устанавливается из вновь созданного архива.

Добавление параметров в подключаемый модуль

А теперь добавим параметр в наш подключаемый модуль. Допустим, требуется, чтобы пользователь мог выбирать появление или сокрытие второго флажка в регистрационной форме. Для реализации такой возможности потребуется следующее.

- Добавить поле параметра в XML-файл подключаемого модуля.
- Ввести в подключаемый модуль логику проверки параметра и удаления его поля, если потребуется.
- Ввести новые языковые ключи в основной файл языковой поддержки.

Поле параметра добавляется в объект класса `JForm` таким же образом, как это делалось раньше из файла `form.xml`. Ниже приведен код разметки элемента `config`, исправленный с учетом добавленного поля параметра.

```
<config>
```

```
  <fields name="params">
    <fieldset name="basic" >
      <field name="show_age_checkbox" type="radio"
            label="PLG_USER_MYREGISTRATION2_SHOW_AGE"
            description="PLG_USER_MYREGISTRATION2_SHOW_AGE_DESC"
            default="0">
         <option value="0">JHIDE</option>
         <option value="1">JSHOW</option>
      </field>
    </fieldset>
  </fields>
</config>
```

Этот код подобен коду разметки параметров шаблона, рассматривавшегося в главе 3. С его помощью варианты выбора вводятся при редактировании данного подключаемого модуля. В этом коде сначала создается элемент разметки `fields` с атрибутом `name`, которому присваивается значение `"params"`, а в нем — элементы разметки `fieldset` и `field` с атрибутами, описывающими поле параметра, причем все они вложены в элемент разметки `config`.

Элемент разметки `fields` с атрибутом `name` должен непременно присутствовать в XML-файле, иначе варианты выбора не появятся на экране редактирования подключаемого модуля. Дело в том, что в файле компоновки, используемом для создания формы редактирования подключаемых модулей (`administrator/components/com_plugins/views/plugin/tmpl/edit_options.php`), осуществляется поиск элемента разметки `fields` со значением `"params"` его атрибута `name`, и при его обнаружении вложенные в него элементы разметки `fieldset` и `field` включаются со всеми своими атрибутами в форму. Если же элемент разметки `fields` отсутствует в XML-файле, варианты выбора не появятся на экране редактирования подключаемого модуля. Это же относится и к компонентам, модулям и шаблонам. Следует, однако, иметь в виду, что элемент разметки `fields` раньше был не нужен при создании файла `form.xml`. А теперь он потребовался нам, поскольку мы добавляем параметры в XML-файл установки для расширения возможностей подключаемого модуля.

В элементе `fieldset` определяется новое поле типа кнопки-переключателя (`radio`) с двумя вариантами выбора: `Hide` (Скрыть) с нулевым значением и `Show` (Показать) с единичным значением. Обратите внимание на то, что эти варианты выбора обозначаются языковыми ключами `JHIDE` и `JSHOW` как отдельные элементы. Подобного рода стандартные языковые ключи повсеместно применяются в Joomla, и поэтому их не нужно вводить в файл языковой поддержки подключаемого модуля. В тех полях, где предоставляется несколько вариантов выбора в виде списка, как, например, в кнопках-переключателях, используются элементы разметки `option` по одному на каждый элемент списка.

Далее необходимо ввести два новых ключа в файл языковой поддержки. Эти ключи вводятся в основной файл `en-GB.plug_user_myregistration2.ini`, а не в системный файл (с расширением `.sys.ini`), поскольку они используются при редактировании отдельного подключаемого модуля в компоненте Plugin Manager. Значение атрибута `label` должно быть коротким, потому что оно будет использоваться для пометки поля. В то же время значение атрибута `description` может быть длинным. Оно будет появляться в подсказке, всплывающей при наведении курсора мыши на поле.

Ниже приведены новые строки, которые должны быть введены в основной файл языковой поддержки подключаемого модуля.

```
PLG_USER_MYREGISTRATION2_SHOW_AGE="Show Age Checkbox"
PLG_USER_MYREGISTRATION2_SHOW_AGE_DESC="Whether to Hide or Show the
↳Show Age check box. If it is shown, it will be required."
```

На рис. 5.8 показаны две символьные строки на избранном языке, отображаемые в форме компонента Plugin Manager. Следует, однако, иметь в виду, что для отображения вокруг текстовых полей сдвоенных знаков звездочки (**), указывающих на правильность перевода текста, необходимо активизировать режим Debug Language по команде Global Configuration⇨System.

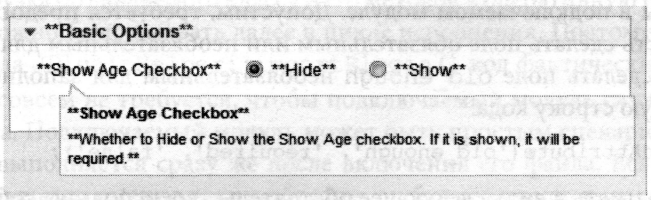


Рис. 5.8. Новый параметр в форме компонента Plugin Manager

И наконец, необходимо внести изменения в исходный код подключаемого модуля, чтобы проверить воздействие на него нового параметра. Ниже приведен исправленный исходный код метода `onContentPrepareForm()`, где внесенные изменения выделены полужирным.

```
public function onContentPrepareForm($form, $data)
{
  // если это не регистрационная форма, проигнорировать ее
  if ($form->getName() != 'com_users.registration') {
      return;
  }

  // загрузить файл языковой поддержки
  $this->loadLanguage();

  // загрузить специальный xml-файл в форму регистрации пользователя
  $form->loadFile(dirname(__FILE__).'/forms/form.xml');
  if (!$this->params->get('show_age_checkbox', '1')) {
     $form->removeField('old_enough');
  }
}
```

Единственным нововведением является блок условного оператора `if` в конце данного метода. В этом блоке сначала получается новый параметр, который затем используется в качестве логического значения при проверке условия в операторе `if`. Так, если выбран вариант `Show`, параметр будет иметь логическое значение `true`. Но поскольку в операторе `if` проверяется обратное условие вследствие применения оператора отрицания (`!`), то строка кода в блоке условного оператора `if` не будет выполняться, если параметр имеет логическое значение `true`. Следовательно, при выборе варианта `Show` вся форма

регистрации пользователя показывается полностью, поскольку кодовый блок условного оператора `if` пропускается.

Если же выбран вариант Hide, кодовый блок условного оператора `if` выполняется, как показано ниже.

```
$form->removeField('old_enough');
```

В приведенной выше строке кода вызывается метод `removeField()` из класса `JForm` для удаления поля `old_enough` из формы. И это все, что нужно сделать. После внесения упомянутых выше изменений проверьте, насколько правильно действует новый параметр.

В классе `JForm` имеются и другие полезные методы, с помощью которых можно видоизменять формы в подключаемом модуле. Допустим, требуется предоставить пользователю возможность сделать поле обязательным или необязательным для заполнения. Так, если требуется сделать поле `old_enough` необязательным для заполнения, достаточно ввести следующую строку кода:

```
$form->setFieldAttribute('old_enough', 'required', 'false');
```

Очень важно иметь в виду следующее обстоятельство: используя событие `onContent PrepareForm`, можно перехватить объект типа `JForm` еще до отображения формы, а затем добавить, удалить или изменить поля формы методами из класса `JForm`. Это позволяет настраивать форму под конкретные потребности.

Применение подключаемых модулей для замещения основных классов

Существующую интегрированную среду подключаемых модулей можно использовать для замещения большинства основных классов в Joomla. Следует, однако, иметь в виду, что дело это непростое, и поэтому браться за него рекомендуется лишь в случае крайней необходимости. Тем не менее знать о такой возможности все же нужно.

Каким образом импортируются подключаемые модули

Как было показано ранее, для запуска подключаемого модуля на выполнение сначала вызывается метод `JPluginHelper::importPlugin()`, импортирующий этот модуль. В итоге классы и методы подключаемого модуля загружаются в рабочую область оперативной памяти. Если проанализировать принцип действия данного метода, то можно обнаружить, что код, выполняющий импорт подключаемого модуля, фактически находится в теле закрытого метода `import()` из класса `JPluginHelper`, находящегося в файле `libraries/joomla/plugin/helper.php`, как показано ниже.

```
if (!isset($paths[$path])) {
    require_once $path;
}
$paths[$path] = true;
```

В первой строке этого фрагмента кода проверяется, введен ли данный конкретный подключаемый модуль. Переменная `$path` представляет собой ассоциативный массив, содержащий все импортированные подключаемые модули. Ключом к этому массиву служит

полный путь к файлу каждого подключаемого модуля, а значением каждого элемента данного массива — имя класса. Для проверки наличия конкретного элемента в массиве вызывается функция `isset()` языка PHP. Если элемент отсутствует в массиве, команда `require_once` языка PHP включает файл искомого подключаемого модуля. И наконец, элементу массива присваивается логическое значение `true`. Этим гарантируется, что в следующий раз данный элемент будет установлен в массиве, а следовательно, выполнять еще раз команду `require_once` для включения того же самого файла не придется.

В отношении рассматриваемого здесь процесса необходимо отметить следующее.

- Как пояснялось ранее, в файлах подключаемых модулей, как правило, объявляются классы, а следовательно, на данном этапе код вообще *не* выполняется. Происходит лишь загрузка объявляемого класса и его методов в оперативную память, чтобы эти методы можно было вызвать далее в цикле исполнения. Поэтому в результате вызова метода `JPluginHelper::importPlugin()` код фактически не выполняется.
- В Joomla совсем *не* требуется, чтобы подключаемый модуль служил для объявления класса. Подключаемый модуль может быть простым сценарием на языке PHP, который выполняется сразу же после включения его файла. Если подключаемый модуль создается именно таким образом, он будет выполнен, как только завершит свою работу метод `JPluginHelper::importPlugin()`, а не в тот момент, когда инициируется событие.

Благодаря этому обеспечивается механизм загрузки любого сценария PHP всякий раз, когда импортируются подключаемые модули.

Каким образом загружаются классы

Далее очень важно понять, каким образом базовые классы Joomla загружаются в рабочую область оперативной памяти. Анализируя функцию `jimport()`, которая обычно применяется для загрузки основных классов Joomla, можно заметить, что она находится в файле `libraries/loader.php` и является совершенно автономной функцией, а не методом класса. Именно поэтому она вызывается по имени функции, а не имени класса. Ниже приведен исходный код этой функции.

```
function jimport($path)
{
    return JLoader::import($path);
}
```

Эта функция просто вызывает метод `JLoader::import()`. Ниже приведена первая строка кода с комментарием из этого метода.

```
// импортировать только библиотеку, если этого еще не было сделано
if (!isset(self::$imported[$key]))
```

В этой строке кода проверяется, был ли класс уже импортирован. В качестве аргумента вызываемой функции `isset()` указывается статический ассоциативный массив `self::$imported[]` с ключом `$key`, совпадающим с аргументом, передаваемым функции `jimport()` (например, `'joomla.plugin.plugin'`), а также логическим значением `true` или `false`. После импорта класса в этот массив добавляется элемент, которому присваивается логическое значение `true`, если импорт произведен успешно, а иначе — логическое

значение false. Таким образом, класс импортируется в Joomla лишь один раз и без повторных попыток. В методах JLoader::load() и JLoader::_autoload() (автозагрузчик платформы) также проверяется, был ли класс уже загружен, прежде чем пытаться загрузить его.

Из всего сказанного выше можно сделать следующий очень важный вывод: если класс уже присутствует в оперативной памяти, т.е. загружен в ее рабочую область, значит, операция его загрузки *должна быть пропущена*, а из метода просто возвращается логическое значение true. Следовательно, ни один из методов Joomla не должен пытаться загрузить класс еще раз.

Это означает, что подключаемый модуль может быть использован для загрузки класса в рабочую область оперативной памяти, при условии, что это будет сделано *до* того, как данный класс загрузится основными программами Joomla. В таком случае методы из загруженного класса будут применяться вместо методов из основного класса. И зачастую системные подключаемые модули загружаются в рабочую область оперативной памяти в самом начале цикла исполнения Joomla, т.е. прежде большинства, но не всех основных классов Joomla.

Пример замещения класса JTableNested

Проиллюстрируем все сказанное выше на кратком и простом примере, заместив класс JTableNested. Этот класс является родительским для всех классов вложенных таблиц в Joomla, например, класса JTableCategories для таблицы #__categories. В данном примере будет показано, каким образом этот класс замещается, а изменение его кода и поведения оставляется на усмотрение читателя. Итак, выполните следующие действия.

1. Создайте новую папку под названием plugins/system/myclasses и скопируйте в нее файл libraries/joomla/database/tablenested.php. В итоге вы получите файл plugins/system/myclasses/tablenested.php. (Напомним, что во все создаваемые папки следует ввести файл index.html.)
2. Отредактируйте новый файл, заменив в нем метод rebuild() приведенным ниже кодом.

```
public function rebuild($parentId = null, $leftId = 0, $level = 0, $path = '')
{
    exit('From myclasses/tabelnested.php file');
}
```

   В этом коде просто подтверждается, что вместо основного класса выполняется класс его замещения. После щелчка на кнопке Rebuild (Перестроить), например, на экране Category Manager: Articles, должен произойти выход из программы с приведенным выше сообщением, если выполняется замещающий вариант метода rebuild().
3. А теперь необходимо ввести подключаемый модуль myclasses, загружающий вместо основного класса замещающий его класс. С этой целью создайте новый файл myclasses.php в папке plugins/system/myclasses.
4. Введите в новом файле (plugins/system/myclasses/myclasses.php) код, приведенный в листинге 5.13.

Листинг 5.13. Исходный код из файла `myclasses.php`

```
<?php
/**
 * Демонстрация подключаемого модуля для замены основного класса.
 * Запускается на выполнение при первом же импорте из системы, т.е.
 * до наступления события onBeforeInitialise.
 */
// прямой доступ отсутствует
defined('_JEXEC') or die;

// заменить основной класс JTableNested вариантом его замещения
include_once JPATH_ROOT.'/plugins/system/myclasses/tablenested.php';
```

Обратите внимание на то, что в этом коде класс не объявляется, а просто выполняется сценарий. Это означает, что данный код будет выполнен сразу же после импорта системных подключаемых модулей, т.е. до наступления первого системного события. В данном коде просто включается новый исходный файл `tablenested.php`.

5. Создайте XML-файл (`plugins/system/myclasses/myclasses.xml`) для данного подключаемого модуля, введя в него код, приведенный листинге 5.14.

Листинг 5.14. Исходный код из файла `myclasses.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<extension version="2.5" type="plugin" group="system">
    <name>plg_system_myclasses</name>
    <author>Mark Dexter and Louis Landry</author>
    <creationDate>January 2012</creationDate>
    <copyright>Copyright (C) 2012 Mark Dexter and Louis Landry.</copyright>
    <license>GPL2</license>
    <authorEmail>admin@joomla.org</authorEmail>
    <authorUrl>www.joomla.org</authorUrl>
    <version>1.6.0</version>
    <description>MyClasses plugin demonstration</description>
    <files>
        <filename plugin="myclasses">myclasses.php</filename>
        <filename>index.html</filename>
    </files>
    <config>
    </config>
</extension>
```

6. Перейдите в административную часть веб-сайта на Joomla, а затем найдите и установите подключаемый модуль с помощью функций Discover и Install соответственно, как и в предыдущих примерах. Не забудьте также активизировать подключаемый модуль в компоненте Plugin Manager.
7. Перейдите по команде Content⇨Category Manager (Содержимое⇨Диспетчер категорий) в административной части веб-сайта на Joomla и щелкните на пиктограмме Rebuild. Выполнение Joomla должно остановиться, а на экране — появиться сообщение "`From myclasses/tabelnested.php file`", указывающее на то, что основной класс успешно замещен.

Такой прием может быть использован для замещения большинства основных классов Joomla, кроме тех, которые уже загружены до импорта системных подключаемых модулей. Если основной класс замещается подобным способом, то беспокоиться о перезаписи этого класса при переходе на новую версию Joomla не приходится. Следовательно, пользоваться таким приемом намного удобнее, чем просто влезать в исходный код базовых файлов с целью его замены. Однако если в любых замещаемых основных классах обнаружены и исправлены программные ошибки, то их придется проверить и исправить самостоятельно и в классах замещения. Это особенно важно для исправления программных ошибок, нарушающих безопасность системы.

Передовой опыт разработки и применения подключаемых модулей

В отношении подключаемых модулей очень важно знать, когда их применять. Это означает, что нужно хорошо разбираться в событиях, инициируемых в Joomla, а также в стандартных режимах работы, которые можно заместить. Как правило, подключаемый модуль может значительно упростить задачу применения отдельного модуля или компонента. Ниже приведены некоторые рекомендации относительно разработки и применения подключаемых модулей.

- Подключаемые модули выполняются в том порядке, в каком они появляются в списке Ordering (Упорядочение) на экране Plugin Manager. Как правило, порядок их расположения особого значения не имеет, кроме тех случаев, когда при наступлении одного и того же события на выполнение запускаются несколько подключаемых модулей, а результаты выполнения одного из них могут оказать влияние на работу другого. В подобных случаях можно изменить порядок расположения подключаемых модулей в упомянутом выше списке, откорректировав тем самым последовательность их выполнения.
- Как правило, подключаемые модули именуются в соответствии с условными обозначениями, принятыми для классов и методов, иначе они не будут вызываться правильно. Исключением из этого правила является ситуация, когда требуется выполнить сценарий при импорте подключаемого модуля. В этом случае имеет значение только имя файла и папки, как в примере замещения класса из предыдущего раздела.
- Для инициирования разных событий требуются разные сигнатуры методов. Поэтому необходимо проверить, какие именно значения для каждого события передаются методу в качестве аргументов и какие значения он должен возвращать, если, конечно, он должен это делать. В информационном хранилище Joomla имеется папка **`tests/plugins`**, в которой содержатся файлы с примерами подключаемых модулей каждого типа. Эти примеры наглядно демонстрируют все события и сигнатуры методов для каждого типа подключаемых модулей. Кроме того, события, обрабатываемые подключаемыми модулями, описаны в документации Joomla по адресу `http://docs.joomla.org/Plugin/Events`.
- Старайтесь выбирать наиболее подходящее событие для обработки подключаемым модулем. Если для этой цели требуется новое событие, например, в специальном

компоненте, просто инициируйте его в нужном месте программы. А если вы считаете, что новое событие должно быть введено в основной класс, проконсультируйтесь с другими разработчиками в списках рассылки по разработке Joomla. Если остальные разработчики согласятся с вами, событие можно будет ввести в основной класс.

Резюме

В этой главе были рассмотрены подключаемые модули. Сначала в ней был сделан краткий обзор некоторых базовых подключаемых модулей, а затем приведены примеры разработки собственных подключаемых модулей, в том числе и для настройки процесса регистрации пользователей на веб-сайте Joomla.

По ходу изложения материала этой главы было показано, что подключаемые модули позволяют сделать немало полезного, если поместить совсем немного кода в нужном месте, а также предоставить больше удобств для работы администратора веб-сайта. И в конце главы было пояснено, каким образом основные классы Joomla замещаются по мере необходимости с помощью подключаемых модулей. В следующей главе будет рассмотрена еще одна важная разновидность расширений Joomla: модули.

6

Расширение Joomla модулями

В этой главе будет показано, каким образом модули действуют в Joomla. Сначала мы рассмотрим взаимосвязь модулей с компонентами, а затем принцип действия основного модуля. После этого мы создадим новый модуль, в котором информация извлекается из базы данных Joomla. Этот модуль будет дополнен рядом параметров и упакован для упрощения его установки. А в конце главы будут даны некоторые рекомендации относительно извлечения наибольшей пользы из модулей.

Что такое модуль

На типичной странице веб-сайта Joomla отображается один компонент и несколько модулей. Так, на начальной странице веб-сайта версии Joomla 2.5 с выборочными данными представлен компонент Featured Articles (Избранные статьи) и пять модулей (поиска, регистрации и трех меню). Как правило, модули невелики, просты и "легковесны", т.е. они не требуют много времени процессора на обработку. В то же время модули предоставляют сведения и навигационные средства для пользователей.

Модули в сравнении с компонентами

Модули зачастую опираются на компоненты. Например, модуль меню (`mod_menu`) извлекает данные из компонента Menu (`com_menus`), а все модули статей, в том числе архивных, с короткими экстренными сообщениями и т.д.) — из компонента Articles (`com_content`). Многие модули только отображают информацию, а иногда — ссылки, но некоторые из них, например, модули регистрации и поиска, — простые формы.

Как правило, модули не требуют взаимодействия с пользователем, в том числе перехода на следующую страницу. Ниже перечислены некоторые примечательные особенности модулей в Joomla.

- Один и тот же модуль может быть отображен на всех или выбранных страницах веб-сайта. Этот режим отображения назначается в компоненте Module Manager (Диспетчер модулей) при создании модуля.
- Отображение информации в модуле может происходить в зависимости от того, что именно отображается в главном компоненте. Так, в модуле Breadcrumbs (Навигационные цепочки) отображается иерархическая структура меню от начальной страницы до текущего пункта меню.

В отличие от компонентов, которые выполняются, исходя из конкретного URL, модули вызываются как неотъемлемая часть встроенного в Joomla цикла обработки. После выполнения и воспроизведения компонента в Joomla обнаруживаются все модули, которые назначены, во-первых, для текущего пункта меню, а во-вторых — для места, определенного в шаблоне. Модули выполняются по очереди, а результаты выполнения каждого из них вносятся в файл `index.php` шаблона по месту нахождения элемента разметки `jdoc:include`, определяющего место расположения каждого модуля. Для разработки модулей, как правило, требуется писать больше кода, чем для подключаемых модулей, тем не менее они создаются просто и понятно.

Краткий обзор базового модуля

Начнем изучение модулей с базового модуля Latest Users (Последние пользователи). Этот модуль отображает список пользователей, недавно зарегистрировавшихся на веб-сайте. Настраивая параметры этого модуля, можно ограничить количество отображаемых пользователей, показывать ссылки на пользователей и отображать сведения, выбираемые из контактной информации и профиля каждого пользователя.

Рассмотрим сначала папки и файлы данного модуля. Модуль Latest Users относится к пользовательской части веб-сайта, и поэтому он находится в подпапке, входящей в папку `modules`. Если бы это был модуль из административной части веб-сайта, он находился бы в папке `administrator/modules`. Модулю Latest Users присвоено внутреннее имя `mod_users_latest`, по которому и называется его папка. Заглянув в эту папку, в ней можно обнаружить файлы, перечисленные в табл. 6.1, разумеется, кроме файла `index.html`. А теперь перейдем к подробному рассмотрению принципа действия модуля Latest Users.

Таблица 6.1. Файлы модуля Latest Users, кроме файла `index.html`

Имя файла	Описание
`tmpl/default.php`	Файл компоновки модуля (выводит результат выполнения модуля в окне браузера)
`helper.php`	Содержит методы сбора данных для модуля
`mod_users_latest.php`	Точка входа при выполнении модуля
`mod_users_latest.xml`	Установочный XML-файл, где определены поля дополнительных параметров модуля

XML-файл модуля

В XML-файле рассматриваемого здесь модуля содержатся сведения о файлах и папках, используемых для установки и удаления данного модуля, а также определения полей для любых параметров модуля. Файл `mod_users_latest.xml` по своему составу очень похож на XML-файлы подключаемых модулей, рассматривавшихся в главе 5, за исключением того, что атрибут `type` элемента разметки `extension` имеет значение `"module"`. В приведенном ниже коде демонстрируется, каким образом элементы разметки `files` и `languages` определяются в файле `mod_users_latest.xml`.

```
<files>
    <filename module="mod_users_latest">mod_users_latest.php</filename>
     <folder>tmpl</folder>
     <filename>helper.php</filename>
     <filename>index.html</filename>
</files>
<languages>
    <language tag="en-GB">en-GB.mod_users_latest.ini</language>
    <language tag="en-GB">en-GB.mod_users_latest.sys.ini</language>
</languages>
```

В этом фрагменте кода определяется каждый файл модуля, кроме тех файлов, которые находятся в подпапке `tmp`. Следует заметить, что определять элемент `filename` в XML-файле совсем не обязательно, хотя ничто не мешает это сделать. С другой стороны, оба файла языковой поддержки определены в данном файле в элементе `languages`. Как пояснялось ранее, это означает, что оба эти файла будут установлены в основной папке языковой поддержки, а не в папке языковой поддержки данного расширения.

В XML-файле рассматриваемого здесь модуля имеется также следующий элемент разметки:

```
<help key="JHELP_EXTENSIONS_MODULE_MANAGER_LATEST_USERS" />
```

В нем определяется ключ для поиска в справочной системе Joomla статьи о данном модуле. А в остальной части данного файла определяются поля параметров модуля. Среди них три поля определяются в основном элементе разметки `fieldset`, а пять других — в дополнительном элементе разметки `fieldset`. Все эти поля определяются таким же образом, как и в шаблонах и подключаемых модулях, рассматривавшихся в предыдущих главах.

Главный файл модуля

Когда модуль выполняется, он загружает исходный PHP-файл, именуемый таким же образом, как и папка, в которой он находится. В данном случае это файл `modules/mod_users_latest/mod_users_latest.php`. Исходный код из этого файла приведен в листинге 6.1.

Листинг 6.1. Исходный код из файла `mod_users_latest.php`

```
<?php
/**
 * @package      Joomla.Site
 * @subpackage   mod_users_latest
```

```
 * @copyright     Copyright (C) 2005 - 2012 Open Source Matters, Inc. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.
txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

// включить самые последние функции только один раз
require_once dirname(__FILE__).'/helper.php';
$shownumber = $params->get('shownumber', 5);
$name = moduserslatestHelper::getUsers($params);
$linknames = $params->get('linknames', 0);
$moduleclass_sfx = htmlspecialchars($params->get('moduleclass_sfx'));
require JModuleHelper::getLayoutPath('mod_users_latest', $params->
↳get('layout', 'default'));
```

Во второй строке приведенного выше кода включается вспомогательный класс данного модуля. Во вспомогательном классе большинства модулей находятся методы, необходимые для воспроизведения модуля. Именно здесь и происходит большая часть процесса обработки.

В следующей строке кода вызывается метод `getUsers()` из вспомогательного класса для сохранения результата в переменной `$linknames`. Обратите внимание на то, что этому методу в качестве аргумента передается переменная `$params`. В данном файле эта переменная нигде не объявляется. Откуда же она берется? Для того чтобы ответить на этот вопрос, нам придется рассмотреть, как и где выполняется рассматриваемый здесь файл. Как пояснялось в главе 3, модули выполняются по очереди, исходя из назначенного для них места в шаблоне. В конечном итоге данный процесс приводит к следующему фрагменту кода из метода `renderModule()` класса `JModuleHelper`, объявляемого в файле `libraries/joomla/application/module/helper.php`:

```
$content = '';
ob_start();
require $path;
$module->content = ob_get_contents().$content;
```

В выделенной полужирным строке кода переменная `$path` содержит полное имя входного файла модуля. В данном случае требуется файл `mod_users_latest.php`. Команда `require` языка PHP немедленно вводит требуемый файл в текущую программу на PHP. В конечном счете это похоже на копирование и вставку требуемого файла в данном месте текущей программы.

Следует заметить, что файл `mod_users_latest.php` содержит обычный сценарий PHP, но не объявление класса. Поэтому строки кода из этого файла выполняются, как только он включается с помощью команды `require`. Любые переменные, оказавшиеся в области действия при выполнении команды `require`, остаются по-прежнему в области действия и при выполнении требуемого сценария. Это означает, что в сценарии из файла `mod_users_latest.php` одни и те же переменные находятся в той области действия, которая имелась в методе `renderModule()` класса `JModuleHelper`, где выполнялась команда `require`.

Как следует из приведенного ниже фрагмента кода, переменная `$params` устанавливается в методе `renderModule()` раньше.

```
// получить параметры метода
$params = new JRegistry;
$params->loadJSON($module->params);
```

Именно отсюда она и берется. Эта переменная содержит ссылку на объект типа `JRegistry`, в котором хранятся дополнительные параметры, введенные при создании модуля в компоненте Module Manager. Они оказываются доступными при вызове вспомогательного метода данного модуля.

В следующей строке кода из листинга 6.1 параметр `linknames` выбирается среди дополнительных параметров модуля и сохраняется в переменной `$linknames`. Эта переменная будет использоваться при выполнении компоновки данного модуля.

Далее следует строка кода, в которой суффикс класса данного модуля получается из объекта, доступного из переменной `$params`. Суффикс класса модуля представляет собой поле, в котором разработчик веб-сайта может ввести текст при создании модуля. Этот суффикс позволяет точно настроить оформление модуля с помощью вложенных таблиц стилей присваиванием особого класса CSS соответствующим элементам разметки модуля в коде HTML. Для дополнительной обработки получаемого значения вызывается встроенная в PHP функция `htmlspecialchars()`, преобразующая символы, имеющие особое значение в коде HTML, например, угловые скобки (**<** и **>**), одинарные и двойные кавычки, амперсанд (**&**), в специальные знаки HTML (****, **>**, **"** и **&**). Этим гарантируется, что неумышленный или злоумышленный код HTML не попадет в результат, выводимый из данного модуля.

И в последней строке рассматриваемого здесь кода выполняется команда `require`, как показано ниже.

```
require JModuleHelper::getLayoutPath('mod_users_latest',
↳$params->get('layout', 'default'));
```

В этой строке кода выполняется несколько действий. Прежде всего в ней выбирается дополнительный параметр компоновки среди параметров данного модуля. Как пояснялось в главе 4, для модулей можно создать альтернативные компоновки и выбрать их в компоненте Module Manager при создании нового экземпляра модуля. Затем в этой строке кода вызывается метод `getLayoutPath()` из класса `JModuleHelper` с указанием имени модуля и компоновки в качестве аргументов. Этот метод возвращает полное имя файла с требуемой компоновкой. Если альтернативная компоновка не указана, выбирается компоновка, используемая по умолчанию из файла `modules/mod_users_latest/tmpl/default.php`. И файл компоновки содержит обычный сценарий PHP, но не объявление класса, поэтому он выполняется сразу же после его включения с помощью команды `require`. Благодаря этому все переменные, оказавшиеся в области действия на данном этапе выполнения программы, остаются по-прежнему в области действия и при выполнении сценария компоновки.

Рассмотрим подробнее данное проектное решение. Файл `mod_users_lastest.php` стал доступным благодаря выполнению команды `require` в классе `JModuleHelper`. А теперь по этой же команде выбирается файл компоновки. На первый взгляд такое проектное решение кажется немного запутанным, тем не менее оно позволяет работать с более

компактными файлами программ, храня отдельные части программы в разных папках. Эти преимущества перевешивают недостатки, связанные с необходимостью отслеживать места включения одних файлов в другие.

Подведем итог всем функциям кода из файла `mod_users_lastest.php`.

- Загрузка вспомогательного класса в рабочую область памяти с целью сделать доступными его методы.
- Вызов метода из вспомогательного класса для получения данных, предназначенных для модуля.
- Включение с помощью команды `require` файла компоновки, фактически отображающего результаты, выводимые из модуля.

Короче говоря, код из этого файла управляет всем процессом выполнения модуля.

Включение файлов вместо вызова методов

В Joomla файлы обычно включаются с помощью команд `require`, `require_once`, `include` или `include_once` языка PHP. Как пояснялось в предыдущей главе, если включаемый файл содержит объявление класса, этот класс загружается в оперативную память, но код, по существу, не выполняется. А если включаемый файл содержит обычный сценарий PHP, т.е. последовательность операторов, кроме объявления класса или определения функции, этот сценарий выполняется немедленно, а *все переменные, находящиеся в области действия при включении файла, оказываются в области действия включаемого файла*.

Это означает, что при включении файла следует очень аккуратно пользоваться именами переменных. Хорошо то, что мы можем пользоваться переменными, определенными ранее в этом файле, если он содержит оператор `include` или `require` (например, переменной `$params` в файле `mod_users_latest.php` рассматриваемого здесь модуля). Но плохо то, что мы можем получить неожиданный результат, если случайно воспользуемся одной и той же переменной как во включенном, так и во включающем файле. В частности, мы можем случайно изменить значение переменной, используемой далее в вызывающей части программы. Следовательно, мы должны очень аккуратно обращаться с переменными во включаемых файлах, не содержащих объявления классов.

Совсем иначе дело обстоит с классами и методами. Если не указано иное, все переменные в классах и методах являются локальными для класса или метода. Поэтому при вызове метода переменные из вызывающей части программы оказываются за пределами области действия в теле данного метода. Если требуется передать информацию из вызывающей части программы вызываемому методу, мы, как правило, указываем переменные в качестве аргументов этого метода. А если требуется возвратить информацию из вызываемого метода обратно в вызывающую часть программы, мы обычно поручаем это самому методу. Но преимущество такого подхода заключается в том, что в теле методов мы вольны пользоваться любыми именами переменных. И не имеет никакого значения, те же ли это самые переменные, что и в вызывающей части программы. Ведь области их действия не перекрываются.

Вспомогательный класс модуля

Перейдем к рассмотрению вспомогательного класса модуля. Именно здесь собирается вся информация для модуля. В листинге 6.2 приведен исходный код метода `getUsers()`, выполняющего сбор этой информации во вспомогательном класса данного модуля.

Листинг 6.2. Исходный код метода `getUsers()` из вспомогательного класса `modUsersLatestHelper`

```
// отсортировать пользователей по дате активизации
static function getUsers($params)
{
    $db = JFactory::getDbo();
    $query = $db->getQuery(true);
    $query->select('a.id, a.name, a.username, a.registerDate');
    $query->order('a.registerDate DESC');
    $query->from('#__users AS a');
    if (!$user->authorise('core.admin') &&
↳$params->get('filter_groups', 0) == 1)
    {
      $groups = $user->getAuthorisedGroups();
      if (empty($groups))
      {
        return array();
      }
      $query->leftJoin('#__user_usergroup_map AS m ON m.user_id = a.id');
      $query->leftJoin('#__usergroups AS ug ON ug.id = m.group_id');
      $query->where('ug.id in (' . implode(',', $groups) . ')');
      $query->where('ug.id <> 1');
    }
    $db->setQuery($query,0,$params->get('shownumber'));
    $result = $db->loadObjectList();
    return (array) $result;
}
```

Этот метод объявляется как статический. Как пояснялось ранее, это означает, что статический метод вызывается по имени его класса, а не объекта. Обращаясь к аналогии из процедурного программирования, статический метод можно сравнить с подпрограммой.

В первой строке кода этого метода создается объект `$db` типа `JDatabase`. Подобного рода объекты применяются в Joomla всякий раз, когда требуется извлечь информацию из базы данных. В следующей строке кода создается новый объект `$query` типа `JDatabaseQuery`. Следует заметить, что объект типа `JDatabase` используется для формирования запроса при вызове метода `$db->getQuery(true)`. Аргумент с логическим значением `true` указывает объекту типа `JDatabase` предоставить новый, а не сделанный последним запрос, если, конечно, таковой имеется.

Более подробно запросы базы данных рассматриваются в главе 11, а до тех пор рассмотрим, каким образом запрос формируется и выполняется в данном конкретном примере. Базы данных организованы в виде последовательного ряда файлов таблиц, где в каждом поле содержится один тип данных. Эти файлы организованы по строкам и столбцам как электронные таблицы. Например, в таблице `#__content` содержатся данные для статей, в каждой ее строке — данные для одной статьи, а каждом столбце — часть данных для всех статей (например, заглавие или категория).

Для того чтобы извлечь информацию из базы данных, воспользуемся запросом `SELECT`. Этот запрос возвращает данные в формате "строка-столбец". Простой запрос `SELECT` имеет следующий формат:

```
SELECT <список столбцов>
FROM <имя таблицы>
```

```
WHERE <условия отбора>
ORDER BY <столбцы для сортировки результатов в заданном порядке>
LIMIT <номер начальной строки>, <число возвращаемых строк>
```

При написании программы в версии Joomla 1.5 приходилось сначала составлять символьную строку, содержавшую точный текст запроса, а затем передавать запрос базе данных. В версии Joomla 1.6 появилась возможность пользоваться классом `JDatabaseQuery`, упрощающим формирование запросов SQL в Joomla. В этом классе учитываются также любые отличия в базах данных.

Например, в версии Joomla 2.5 поддерживаются базы данных Microsoft SQL Server и MySQL, а в ближайшей перспективе — PostgreSQL. Пользуясь классом `JDatabaseQuery` для формирования всех запросов, мы можем быть уверены в том, что наш код будет выполняться корректно при обращении к этим новым базам данных, поскольку в данном классе автоматически учитываются отличия в командах SQL, применяемых в каждой из них.

Методы из класса `JDatabaseQuery` довольно точно отражают формат команд SQL. Но, в отличие от команд SQL, где отдельные части запроса должны быть записаны в строгом порядке, в классе `JDatabaseQuery` можно формировать запросы базы данных в любом порядке. В приведенном ниже вызове метода `$query->select()` столбцы добавляются в часть `SELECT` формируемого запроса.

```
$query->select('a.id, a.name, a.username, a.registerDate');
```

Здесь из таблицы базы данных выбираются четыре столбца: `id` (идентификатор), `name` (имя), `username` (имя пользователя), а также `registerDate` (дата регистрации). Префикс `a.`, предшествующий имени каждого столбца, обозначает сокращенное имя таблицы (на тот случай, если в запросе указывается несколько таблиц). Сокращенное имя таблицы определяется в методе `from()`.

В следующем вызове метода

```
query->order('a.registerDate DESC');
```

добавляются столбцы, которые будут использоваться в базе данных для сортировки результатов запроса. В данном случае результаты запроса сортируются по дате регистрации пользователей на веб-сайте (`a.registerDate`), причем в убывающем порядке, на что указывает слово `DESC` после имени столбца. Это слово предписывает базе данных сортировать извлекаемые из нее данные в обратном, убывающем (`DESCENDING`) порядке. Напомним, что среди новых пользователей веб-сайта должны быть сначала показаны те, кто активизированы последними. Даты хранятся в базе данных как числа, а следовательно, последние даты представлены большими числами, чем старые даты. Поэтому сортировка по дате регистрации осуществляется в обратном порядке для составления списка пользователей, начиная с зарегистрировавшихся позже всех и кончая зарегистрировавшимися раньше всех.

В следующем далее вызове метода

```
$query->from('#__users AS a');
```

указывается таблица, используемая в запросе. В данном случае происходит обращение к таблице `#__users`, обозначаемой в запросе как `a`. При установке Joomla вы можете выбрать буквы в качестве префиксов для обозначения таблиц базы данных. Эти префиксы затем указываются перед именами таблиц в базе данных. Указывая имена таблиц

в запросе SQL, мы заранее не знаем, какой именно префикс будет выбран для них на отдельном веб-сайте. Именно поэтому в запросах рекомендуется всегда указывать префикс #__ (знак денежной единицы и два последующих знака подчеркивания). Непосредственно перед отправкой запроса в базу данных префикс #__ автоматически преобразуется средствами Joomla в конкретный префикс таблицы, выбранный для каждого веб-сайта на Joomla. В этой книге мы обращаемся к таблицам базы данных, используя префикс #__, но префикс для конкретного веб-сайта на Joomla может быть другим. Слово AS указывает базе данных сокращенное имя a таблицы. Следовательно, для обращения к любому столбцу таблицы #__users можно воспользоваться выражением a. + <имя столбца>.

После строки кода с вызовом метода $query->from() получается объект пользователя и начинается выполнение кодового блока условного оператора if. В этом блоке реализуется возможность фильтровать текущих пользователей по их группам. Если такая возможность (под названием filter_groups) активизирована и текущий пользователь не является привилегированным, т.е. не имеет полномочия администратора (core.admin), то код в данном блоке может быть выполнен.

В этом кодовом блоке сначала получается массив групп, членами которых является текущий пользователь, для чего вызывается метод getAuthorisedGroups() из класса JUser. Если этот массив оказывается пустым, обработка завершается и просто возвращается пустой массив. В противном случае в формируемый запрос вводятся два соединения (с помощью метода leftJoin()) и два оператора WHERE (с помощью метода where()). Более подробно соединения и операторы WHERE рассматриваются в главе 11, а в данном случае соединения позволяют выяснить, каким группам принадлежит текущий пользователь, тогда как операторы WHERE — накладывают на запрос ограничение, чтобы показывать только тех пользователей, у которых хотя бы одна общая группа, кроме группы **1**, которая и так является общей.

После кодового блока условного оператора if следует приведенная ниже строка кода.

```
$db->setQuery($query,0,$params->get('shownumber'));
```

Именно здесь и передается запрос в базу данных, а также указывается номер начальной строки и максимальное число столбцов для запроса. Первым аргументом метода setQuery() в данном примере служит объект $query. Анализируя исходный код метода setQuery() из класса JDatabase, находящегося в файле libraries/joomla/database/database.php, можно заметить, что первым его аргументом является символьная строка. Как упоминалось в главе 4, если в коде PHP объект используется там, где ожидается символьная строка, то вызывается специальный метод __toString() этого объекта, преобразующий его в символьную строку. В классе JDatabaseQuery (из файла libraries/joomla/database/databasequery.php) можно обнаружить метод __toString(), преобразующий объект в действительную команду SQL. А поскольку в качестве первого аргумента метода setQuery() ожидается символьная строка, то при желании можно передать ему не только объект, но и строку с точным текстом запроса. Тем не менее при формировании запросов для Joomla рекомендуется всегда пользоваться методами из класса JDatabaseQuery.

Далее следует приведенная ниже строка кода.

```
$result = $db->loadObjectList();
```

В этой строке кода фактически выполняется сформированный запрос, а возвращаемые его результаты сохраняются в переменной `$result`. В частности, вызываемый метод `loadObjectList()` возвращает список строк и столбцов в виде массива объектов. Каждая возвращаемая строка является стандартным объектом, содержащим поле для каждого столбца, указанного в запросе. В данном примере это поля `id`, `name`, `username` и `register date`. Значения в этих полях выбираются из базы данных по заданной строке. Максимальное число строк определяется значением параметра `shownumber`. В зависимости от конкретной даты мы можем получить от нуля до любого числа строк, но не больше того, что ограничивается значением параметра `shownumber`.

Файл исходной компоновки

В листинге 6.3 приведен код исходной компоновки из файла `default.php`.

Листинг 6.3. Код исходной компоновки из файла `default.php`

```
<?php
/**
 * @package       Joomla.Site
 * @subpackage    mod_users_latest
 * @copyright     Copyright (C) 2005 - 2012 Open Source Matters, Inc. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */
// прямой доступ отсутствует
defined('_JEXEC') or die;
?>
<?php if (!empty($names)) : ?>
    <ul class="latestusers<?php echo $moduleclass_sfx ?>" >
    <?php foreach($names as $name) : ?>
      <li>
        <?php echo $name->username; ?>
      </li>
    <?php endforeach; ?>
    </ul>
<?php endif; ?>
```

Этот простой пример компоновки еще раз наглядно демонстрирует возможности сочетания кода PHP и HTML в одном файле. После оператора `defined()` компоновка начинается с условного оператора `if`. Обратите внимание на применение альтернативного синтаксиса языка PHP в условном операторе `if`. Именно такой синтаксис рекомендуется всегда использовать в файлах компоновки, где сочетаются элементы разметки PHP и HTML. В условном операторе `if` проверяется, не является ли пустым массив `$names`. Это делается для того, чтобы исключить возможность отображения пустого элемента `ul`.

А откуда берется переменная `$names`? Напомним, что файл текущей компоновки выполняется с помощью команды `require` из файла `mod_users_latest.php`. Как пояснялось ранее, сценарий из этого файла включается, а не вызывается как метод из класса, и поэтому все переменные, оказавшиеся в области действия при включении файла этого сценария, по-прежнему остаются в области действия при его выполнении. Это все равно, что вставить файл текущей компоновки в другой файл (в данном случае `mod_users_latest.`

php) там, где он был включен. Следовательно, переменная $names берется из следующей строки кода в файле mod_users_latest.php:

```
$names = moduserslatestHelper::getUsers($params);
```

В следующей строке кода из файла исходной компоновки создается элемент ul разметки HTML. В него вставляется переменная $moduleclass_sfx в качестве атрибута класса данного элемента. В рассматриваемой здесь строке кода демонстрируется очень важное положение относительно употребления пробелов в исходных файлах PHP. Как правило, пробелы в исходных файлах PHP особого значения не имеют. Так, приведенные ниже выражения совершенно равнозначны.

```
$x = $y + $z;
```

и

```
$x=$y+$z;
```

Но в строке кода

```
<ul class="latestusers<?php echo $moduleclass_sfx ?>" >
```

команда echo используется для вставки переменной в атрибут элемента разметки ul. В этом случае пробелы имеют такое же значение, как и при наборе буквального текста. Между словом latestusers и значением, возвращаемым командой echo, пробелы отсутствуют, и поэтому данное значение присоединяется к слову latestusers *без пробелов*. Так, если бы переменная $moduleclass_sfx имела значение "my_suffix", то выполнение команды echo дало бы следующий результат:

```
<ul class="latestusersmy_suffix" >
```

А если бы потребовался суффикс класса данного модуля для создания еще одного класса CSS, что зачастую и делается, нам пришлось бы ввести этот суффикс в форму с начальным пробелом. Благодаря этому начальный пробел отделяет имя стандартного класса от его суффикса. Так, если бы ввести <space>my_suffix в качестве суффикса класса, то элемент разметки ul, был бы воспроизведен следующим образом:

```
<ul class="latestusers my_suffix" >
```

и этот элемент можно было бы оформить в стиле из класса CSS с суффиксом my_suffix.

Далее следует фрагмент кода с циклом foreach по переменной $names. Цикл foreach очень удобен для поочередного обращения ко всем элементам массива. Кодовый блок в теле этого цикла выполняется по одному разу для каждого элемента массива. В этом кодовом блоке переменная $name содержит текущий элемент массива. Если же массив пуст, весь этот кодовый блок пропускается.

В теле цикла foreach создается элемент разметки li, в который вставляется имя пользователя без элемента привязки. И завершается код исходной компоновки закрытием элементов разметки HTML в том же порядке, в каком они открывались. Сначала закрывается элемент разметки li, затем элемент разметки с вложенным циклом foreach, далее элемент разметки ul и наконец элемент разметки с вложенным условным оператором if. Обратите внимание, насколько отступы облегчают чтение кода. Они наглядно показывают не только порядок вложения одних кодовых блоков или элементов разметки в другие, но и всю структуру кода HTML, что особенно важно при сочетании элементов разметки PHP и HTML.

Отображение статей текущего автора

Опираясь на полученные знания, попробуем создать свой собственный модуль. Этот модуль будет похож на основной модуль Articles — Related Articles (Статьи — Связанные статьи), отображающий список статей с одним или несколькими ключевыми словами, общими для них и текущей статьи, выводимой на странице. А наш модуль будет отображать статьи, у которых тот же самый автор, что и у текущей статьи на странице. У этого модуля будут также параметры, позволяющие администратору веб-сайта определять количество статей и порядок их отображения.

Разрабатывая расширения, которые могут быть использованы другими, мы должны присвоить им такие имена, чтобы они стали неповторимыми среди разработчиков Joomla. И сделать это проще всего, введя, например, название своей организации в имя расширения. Так, если взять за основу название этой книги, то такую организацию можно было бы назвать "Joomla Programming Book Examples" (Примеры из книги по программированию для Joomla), а сокращенно — "joompro". Исходя из этого, нашему новому модулю нетрудно присвоить имя `mod_joompro_articles_author`.

Структура модуля

Файлы, требующиеся для нашего модуля, приведены на рис. 6.1.

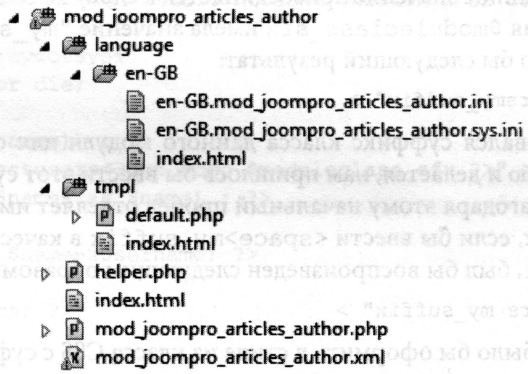


Рис. 6.1. Файлы, требующиеся для примера создаваемого модуля

Список этих файлов похож на список файлов рассмотренного ранее модуля Latest Users. Но в данном случае файлы языковой поддержки помещены в папку самого модуля, а не в основную папку языковой поддержки.

В версии Joomla 1.6 была внедрена логика, позволяющая использовать файлы языковой поддержки в папках самих расширений, помимо файлов, находящихся в основных папках языковой поддержки (`language` и `administrator/language`). И если файл языковой поддержки не будет найден в основной папке, его поиск будет продолжен в папке расширения. Это позволяет хранить файлы языковой поддержки расширения отдельно от базовых файлов языковой поддержки.

XML-файл модуля

Итак, начнем разработку модуля с XML-файла. Этот файл выполняет несколько функций. Так, у элемента разметки `extension` имеются атрибуты для указания типа (`type`), версии (`version`), клиента (`client`); т.е. администратора веб-сайта и метода (`method`) разрабатываемого модуля. Эти атрибуты используются установщиком расширений Joomla в процессе их установки. В элементе разметки `files` предоставляется список файлов и папок (иначе называемый декларацией), откуда установщик расширений Joomla может выяснить, какие именно файлы и папки относятся к устанавливаемому расширению. Это дает возможность копировать файлы в процессе установки и удалять их по ходу удаления самого модуля.

В элемент разметки `extension` вложены также элементы, предоставляющие в описательной форме сведения о модуле, в том числе его название, автора, авторское право и т.д. И наконец, XML-файл может содержать элемент разметки `config`, определяющий любые параметры модуля, появляющиеся в компоненте Module Manager.

Первая часть исходного кода из XML-файла разрабатываемого модуля приведена в листинге 6.4.

Листинг 6.4. Исходный код из файла `mod_joompro_articles_author.xml`, часть 1

```
<?xml version="1.0" encoding="utf-8"?>
<extension
    type="module"
    version="2.5.0"
    client="site"
    method="upgrade">
    <name>mod_joompro_articles_author</name>
    <author>Mark Dexter and Louis Landry</author>
    <creationDate>January 2012</creationDate>
    <copyright>Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↳reserved.</copyright>
    <license>GNU General Public License version 2 or later; see
⇨LICENSE.txt</license>
    <authorEmail>admin@joomla.org</authorEmail>
    <authorUrl>www.joomla.org</authorUrl>
    <version>2.5.0</version>
    <description>MOD_JOOMPRO_ARTICLES_AUTHOR_DESCRIPTION</description>
    <files>
        <filename module="mod_joompro_articles_author">
↳mod_joompro_articles_author.php</filename>
        <folder>tmpl</folder>
        <filename>helper.php</filename>
        <filename>index.html</filename>
        <filename>mod_joompro_articles_author.xml</filename>
        <folder>language</folder>
    </files>
```

Этот код похож на код из рассматривавшегося ранее файла `mod_users_latest module.xml`. В частности, элемент разметки `name` обозначает имя папки, а также имя главного PHP-файла и самого XML-файла данного модуля. А в системном файле языковой поддержки с расширением `.sys.ini` находится ключ для перевода этого имени в

нечто более удобочитаемое и описательное, когда модуль появляется в компоненте Module Manager. Пример такого ключа приведен ниже.

```
MOD_JOOMPRO_ARTICLES_AUTHOR="Articles by Current Author"
```

Этот ключ указан также в элементе разметки `description`, предоставляя тем самым описание данного модуля. Ключ описания модуля определяется в обоих файлах языковой поддержки.

Элемент разметки `files` мало чем отличается своим содержимым от аналогичного элемента разметки модуля `mod_users_latest`. У первого вложенного в него элемента разметки `filename` имеется атрибут `module`, определяющий имя модуля и указывающий файл для загрузки при выполнении модуля. Как и в модуле `mod_users_latest`, в данном случае указывается папка `tmpl`, куда помещается файл компоновки модуля. Затем перечисляются файлы `helper.php`, `index.html` и сам XML-файл. Напомним, что перечислять XML-файл совсем не обязательно, но все же желательно.

Отличие данного примера модуля от предыдущего заключается, в частности, в том, что в рассматриваемый здесь XML-файл добавлен элемент разметки `folder`, в котором указывается папка языковой поддержки модуля. В этой папке будет находиться подпапка с двумя файлами, имеющими расширение `.ini`, для каждого языка, включаемого вместе с модулем.

Вторая часть исходного кода из XML-файла разрабатываемого модуля приведена в листинге 6.5.

Листинг 6.5. Исходный код из файла `mod_joompro_articles_author.xml`, часть 2

```
  <config>
    <fields name="params">
      <fieldset name="basic">
        <field
          name="count"
          type="text"
          default="5"
          label="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_LABEL"
          description="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_DESC">
        </field>
        <field name="article_ordering" type="list"
          validate="options"
          default="a.title"
          label="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_ARTICLEORDERING_LABEL"
          description=
↳"MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_ARTICLEORDERING_DESC">
          <option value="a.title">JGLOBAL_TITLE
          </option>
          <option value="a.hits">MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_HITS_VALUE
         </option>
          <option value="a.created">
↳MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_CREATED_VALUE
          </option>
          <option value="a.publish_up">
↳MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_STARTPUBLISHING_VALUE
          </option>
```

```
        </field>
        <field name="article_ordering_direction" type="list"
          validate="options"
          default="ASC"
          label="MOD_ARTICLES_CATEGORY_FIELD_ARTICLEORDERINGDIR_LABEL"
          description="MOD_ARTICLES_CATEGORY_FIELD_ARTICLEORDERINGDIR_DESC">
          <option value="DESC">
↳MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_DESCENDING_VALUE
          </option>
          <option value="ASC">MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_ASCENDING_
VALUE
          </option>
        </field>
      </fieldset>
```

В этой части XML-файла определяются специальные параметры разрабатываемого модуля. Таких параметров три. Первый из них называется `count` и служит для ввода количества статей, которые требуется отобразить. По умолчанию это количество равно **5**, но пользователь может указать любое количество статьей. Обратите внимание на то, каким образом этот параметр устанавливается: от пользователя не требуется вводить действительное целое значение; он может ввести и текст. Далее в этой главе будут рассмотрены некоторые подходы к улучшению проверки достоверности в поле данного параметра.

Следующий параметр, `article_ordering`, определяет порядок расположения статей в списке. Для этого предоставляются четыре варианта выбора: по названию, количеству обращений, дате написания и дате публикации. Каждый элемент разметки `option` содержит атрибут `value` и соответствующее текстовое описание варианта выбора. Значение атрибута `value` будет передаваться форме при выборе соответствующего варианта. Текст этого значения снабжен языковым ключом, по которому выбирается соответствующее значение из файла (с расширением `.ini`) языковой поддержки данного модуля для последующего перевода. Языковые ключи для текстовых описаний вариантов выбора получились довольно длинными, поскольку в них входит имя данного модуля. Но, несмотря на всю свою громоздкость, они устраняют конфликты по совпадению с именами в других файлах языковой поддержки, которые могут быть загружены одновременно.

В каждом атрибуте `value` указывается точное имя столбца таблицы из базы данных, который будет использоваться в операторе `ORDER BY` запроса SQL для сортировки списка статей. Помимо имени столбца, указывается идентификатор `a.` данной таблицы. А при формировании запроса следует указать имя таблицы с помощью оператора `FROM` со словом `AS a`, как показано ниже.

```
$query->from('#__content AS a');
```

Таким образом, значение, выбираемое из параметра `article_ordering`, может быть использовано непосредственно в запросе базы данных.

И последним в листинге 6.5 является параметр `article_ordering_direction`, определяющий порядок сортировки статей в списке. Он допускает установку одного из двух значений по выбору: `ASC` или `DESC`. Выбранное значение вставляется непосредственно в запрос базы данных и, в частности, в оператор `ORDER BY`, чтобы указать порядок сортировки результатов запроса по нарастающей (`ASC`), как обычно, или же по убывающей (`DESC`).

Следует заметить, что в разметку обоих параметров, `article_ordering` и `article_ordering_direction`, включен атрибут `validate="options"` для проверки соответствия значений этих параметров, введенных в передаваемой форме, тем, что указаны в списке. Это очень важная мера предосторожности, не позволяющая злоумышленнику передавать недостоверные данные в форме. Мы еще вернемся к данному вопросу далее в этой главе.

И последняя часть исходного кода из XML-файла разрабатываемого модуля приведена в листинге 6.6.

Листинг 6.6. Исходный код из файла `mod_joompro_articles_author.xml`, часть 3

```
      <fieldset
        name="advanced">
         <field
           name="layout"
           type="modulelayout"
           label="JFIELD_ALT_LAYOUT_LABEL"
           description="JFIELD_ALT_MODULE_LAYOUT_DESC" />

         <field
           name="moduleclass_sfx"
           type="text"
           label="COM_MODULES_FIELD_MODULECLASS_SFX_LABEL"
           description="COM_MODULES_FIELD_MODULECLASS_SFX_DESC" />

         <field
           name="cache"
           type="list"
           default="1"
           label="COM_MODULES_FIELD_CACHING_LABEL"
           description="COM_MODULES_FIELD_CACHING_DESC">
           <option
             value="1">JGLOBAL_USE_GLOBAL</option>
           <option
            value="0">COM_MODULES_FIELD_VALUE_NOCACHING</option>
         </field>
         <field
           name="cache_time"
           type="text"
           default="900"
           label="COM_MODULES_FIELD_CACHE_TIME_LABEL"
           description="COM_MODULES_FIELD_CACHE_TIME_DESC" />
         <field
            name="cachemode"
           type="hidden"
           default="static">
           <option
             value="static"></option>
         </field>
       </fieldset>
     </fields>
   </config>
</extension>
```

Эта часть кода является точной копией дополнительного элемента `fieldset` из XML-файла разметки любого другого модуля в пользовательской части веб-сайта (например, `mod_articles_archive.xml`). В ней определяются следующие стандартные параметры модуля.

- Компоновка — имя файла дополнительной альтернативной компоновки, как пояснялось в главе 4.
- Суффикс класса модуля — это суффикс дополнительного класса CSS стилевого оформления отдельных экземпляров или модулей.
- Кеширование — активизация или отмена режима кеширования для данного экземпляра или модуля.
- Время кеширования — период времени в секундах, отводимый для использования кешированнной копии модуля.

Четвертый параметр, `cachemode`, демонстрирует очень важное свойство полей в классе `JForm`. Поле этого параметра содержит значение атрибута `type="hidden"`, обозначающее скрытый тип. Это означает, что данное поле будет сохранено как и любое другое поле. Но поскольку оно скрыто, то пользователь вообще не увидит его, а следовательно, не сможет его изменить. Это поле воспроизводится как элемент `input` разметки HTML с атрибутом `type="hidden"`, т.е. как скрытое.



Как пояснялось ранее, не следует забывать, что злоумышленники все же могут манипулировать скрытыми элементами формы, используя, например, такое дополнение веб-браузера, как Firebug. Поэтому и скрытые поля нужно всегда проверять на достоверность. Это следует делать из соображений безопасности, а не только ради удобства пользовательского интерфейса. Иными словами, используя какое-нибудь значение в коде, мы должны принять такие же меры защиты против появления неожиданных или злоумышленных значений в скрытых полях, как и в обычных полях формы.

Исходный файл PHP

В листинге 6.7 приведен исходный код из файла `mod_joompro_articles_author.php` разрабатываемого модуля.

Листинг 6.7. Исходный код из файла `mod_joompro_articles_author.php`

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.
txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

// включать функции объединения только один раз
JLoader::register('modJoomProArticlesAuthorHelper',
↳dirname(__FILE__).'/helper.php');
```

```
$list = modJoomProArticlesAuthorHelper::getList($params);
// показывать модуль только.в том случае, если есть что показывать
if (isset($list[0])) {
    $moduleclass_sfx = htmlspecialchars($params- >get('moduleclass_sfx'));
    require JModuleHelper::getLayoutPath('mod_joompro_articles_author',
↳$params->get('layout', 'default'));
}
```

Во второй строке приведенного выше кода вызывается метод `JLoader::register()` для загрузки вспомогательного класса, хотя это можно было бы с тем же успехом сделать и с помощью команды `require_once`. Но в методе `JLoader::register()` используется средство автозагрузки на языке PHP, да и выполняется он быстрее, чем команда `require_once`. Поэтому там, где это возможно и уместно, следует пользоваться методом `JLoader::register()` (или командой `jimport`, которая, в свою очередь, вызывает данный метод) вместо команды `require_once`.

В следующей строке кода вызывается вспомогательный класс для получения списка статей. А далее проверяется, есть ли что-нибудь для отображения. Для того чтобы проверить, есть ли в массиве `$list` какое-нибудь содержимое, вместо условного выражения

```
if (isset($list[0])) {
```

можно было бы воспользоваться выражением

```
if (count($list)) {
```

Но в данном случае используется функция `isset()`, поскольку она действует быстрее. А в функции `count()` осуществляется циклическое обращение ко всем элементам массива с целью подсчитать их количество. Но нас не интересует, имеется ли в массиве один или сто элементов; нам нужно выяснить, есть ли в нем хотя бы один элемент. А поскольку это индексированный, а не ассоциативный массив, то при наличии в нем какого-нибудь содержимого у него должен быть хотя бы первый элемент с нулевым индексом (`$list[0]`).

Если имеется что-нибудь для отображения, то далее из соответствующего параметра выбирается суффикс класса данного модуля, а затем включается файл компоновки. Для включения файла компоновки используется команда `require`, поскольку это не класс, тогда как упоминавшийся ранее метод `JLoader::register()` служит лишь для загрузки классов. Для этой цели не подходит и команда `require_once`, поскольку и так известно, что файл компоновки включается в данном месте кода лишь один раз. А назначение команды `require_once` состоит в том, чтобы исключить любые попытки объявить тот же самый класс или функцию дважды, что привело бы к ошибке в коде PHP. Кроме того, команда `require_once` проверяет, включен ли уже файл, и поэтому она выполняется медленнее, чем команда `require`. Следовательно, командой `require_once` не стоит пользоваться там, где это неуместно.

Для получения имени файла компоновки вызывается метод `JModuleHelper::getLayoutPath()`. Это дает пользователю возможность составлять и применять в модулях альтернативные компоновки, а также замещения компоновки шаблонов.

В рассматриваемом здесь коде было принято незначительное на первый взгляд, но очень важное проектное решение: проверить, содержит ли что-нибудь переменная `$list`, прежде чем обращаться к компоновке. Это означает, что если ни одной статьи текущего автора не обнаружено, то ничего и не выводится. Если бы в отсутствие статей текущего

автора соответствующее сообщение могло бы выводиться из компоновки, то обращаться к ней можно было бы и при наличии пустого списка статей.

Вспомогательный файл

В листинге 6.8 приведена первая часть вспомогательного файла PHP с объявлением класса `modJoomProArticlesAuthorHelper`.

Листинг 6.8. Исходный код из файла `helper.php`, часть 1

```
<?php
/**
* @copyright            Copyright (C) 2012 Mark Dexter and Louis Landry.
↳ All rights reserved.
 * @license       GNU General Public License version 2 or later;
↳ see LICENSE.txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

JLoader::register('ContentHelperRoute',
↳JPATH_SITE.'/components/com_content/helpers/route.php');

abstract class modJoomProArticlesAuthorHelper
{
    public static function getList(&$params)
    {
        // инициализировать возвращаемую переменную
        $items = array();

        // обработать только в том случае, если это
        // представление для единственной статьи
        $option = JRequest::getCmd('option');
        $view = JRequest::getCmd('view');
        if ($option == 'com_content' && $view == 'article')
        {
```

В этом файле сначала регистрируется метод из класса `ContentHelperRoute`, который будет использоваться в дальнейшем, а затем объявляется абстрактный класс. Возможность создавать абстрактные классы была внедрена в версии языка PHP 5. Объект абстрактного класса создать нельзя. Как правило, абстрактные классы применяются в следующих целях.

- Для создания класса с одним или более статическим методом.
- Для создания класса, предназначенного исключительно в качестве прототипа для подклассов (характерным тому примером служит класс `JDatabase` из файла `libraries/joomla/database/database.php`).

В данном случае у абстрактного класса будет лишь один статический метод `getList()`. Он принимает в качестве аргумента переменную `&$params` со ссылкой на объект типа `JRegistry`, содержащий введенные параметры данного модуля.

В первой строке кода из тела данного метода переменная `$items` инициализируется пустым массивом. Инициализировать переменные в языке PHP совсем не обязательно. Но если попытаться воспользоваться неинициализированной переменной, то в конечном итоге будет получено сообщение, называемое в терминологии языка PHP *уведомляющим*. Поэтому настоятельно рекомендуется инициализировать все переменные, задав их начальные значения, прежде чем пользоваться ими. Если режим извещения об ошибках в среде разработки настроен на максимальную чувствительность, то выводимые уведомляющие сообщения языка PHP позволят отловить типичные программные ошибки в процессе разработки.

Одной из частых причин появления уведомляющих сообщений языка PHP служит установка переменной только в кодовом блоке условных операторов `if/then`. Если этот блок пропускается из-за того, что проверяемое условие оказывается ложным, то переменная в нем вообще не устанавливается. И если она используется в дальнейшем, то появляется сообщение языка PHP, уведомляющее о том, что она не инициализирована. Поэтому рекомендуется выработать в себе хорошую привычку инициализировать переменные, используемые в кодовых блоках условных операторов `if/then`. Это же относится и к другим кодовым блокам, включая циклы, поскольку они могут быть пропущены во время выполнения.

В двух следующих строках обе переменные инициализируются данными, полученными из запроса. В частности, метод `JRequest::getCmd()` вызывается для получения значений элементов `option` и `view` из массива запроса. Метод `getCmd()` отфильтровывает запрос, чтобы он содержал только буквы, числа, дефисы и знаки подчеркивания. Этим предотвращаются всякие попытки внедрить в переменные злонамеренный код (например, команды JavaScript или SQL).

Следующая строка кода выполняет очень важную функцию. В рассматриваемом здесь модуле статьи одного и того же автора отображаются в виде текущей статьи на странице. Но ведь на многих страницах веб-сайта статьи могут вообще не появляться. В частности, на странице может появиться другой компонент, например, контактная информация или веб-ссылки, но не исключено, что и список статей. В таком случае данному модулю нечего будет отображать. Поэтому, прежде чем приступать к составлению списка статей, необходимо убедиться в том, что мы находимся в представлении для единственной статьи, что, собственно, и делается в рассматриваемой здесь строке кода. Представление для единственной статьи определяется значением `"com_content"` элемента разметки `option` и значением `"article"` элемента разметки `view`. Эти значения появятся в URL каждой страницы с единственной статьей, если установить кнопку-переключатель No параметра Search Engine Friendly URLs (Удобные для поиска URL) на вкладке SEO (Оптимизация поискового механизма) диалогового окна Global Configuration (Глобальная настройка). Но в любом случае эти значения устанавливаются в переменной `$_REQUEST`.

А теперь, когда известно, что тип пункта меню выбран правильно, можно приступать к составлению списка статей. В листинге 6.9 приведена вторая часть вспомогательного PHP-файла.

Листинг 6.9. Исходный код из файла `helper.php`, часть 2

```
// получить объект базы данных
$db = JFactory::getDbo();
$app = JFactory::getApplication();
```

```
$user = JFactory::getUser();
$id = JRequest::getInt('id');

// получить уровни доступа в виде разделенной
// запятыми символьной строки
$levels = implode(',', $user->getAuthorisedViewLevels());

// установить в полях текущую и пустую даты
$date = JFactory::getDate();
$now = $date->toSql();
$nullDate = $db->getNullDate();

// инициализировать объект запроса
$query = $db->getQuery(true);

// запросить базу данных с целью получить сведения
// об авторе текущей статьи
$query->select('id, created_by, created_by_alias');
$query->from('#__content');
$query->where('id = ' . (int) $id);
$db->setQuery($query);
$currentArticle = $db->loadObject();
```

В этой части исходного кода из вспомогательного файла PHP выполняются два запроса. По первому запросу получается автор текущей статьи, а по второму — осуществляется поиск других статей этого же автора во всей таблице базы данных.

В трех первых строках кода объекты типа `JDatabase`, `JApplication` и `JUser` получаются с помощью методов из класса `JFactory`. Затем идентификатор из запроса получается с помощью метода `getInt()`. Этот метод обеспечивает получение целочисленного значения, отфильтровывая любое злоумышленное или непредвиденное содержимое в запросе.

В следующей строке кода получается список уровней доступа, разрешенных текущему пользователю для просмотра информации на веб-сайте. Для этого вызывается метод `getAuthorisedViewLevels()`, возвращающий массив уровней доступа для всех групп, которым принадлежит данный пользователя. С помощью функции `implode()` языка PHP этот массив преобразуется в разделенный запятыми список.

Далее необходимо принять проектное решение относительно того, как быть в том случае, если имеются неопубликованные статьи или те статьи, к которым у текущего пользователя нет доступа. Для этого было бы проще всего сформировать такой запрос, чтобы текущему пользователю были доступны для просмотра только опубликованные статьи. С другой стороны, можно было бы включить в запрос те статьи, которые недоступны другим пользователям, предоставить текущему пользователю ссылку для регистрации, прежде чем получить возможность просматривать статьи с ограниченным доступом. В данном примере мы выберем первое, более простое решение, ограничив результаты запроса теми статьями, которые действительно можно просматривать.

В следующих трех строках кода обрабатываются даты, которые будут использоваться в запросах. В Joomla статьи находятся в определенных, устанавливаемых датами, временны*х* рамках, в течение которых они публикуются. Поэтому в запрос следует включить статьи с текущей датой, находящейся в пределах между начальной и конечной датами публикации. Но на практике дело обстоит сложнее, поскольку в столбцах таблицы могут оказаться и

пустые даты, означающие неопределенный начальный или конечный срок публикации. Более того, в различных базах данных применяются разные форматы как для текущей, так и для пустой даты.

В первой из рассматриваемых здесь строк кода средствами класса `JFactory` получается объект текущих даты и времени из среды Joomla. Затем переменная `$now` создается в том формате, который требуется для базы данных. Для этого вызывается метод `toSql()` из класса `JDate`. И наконец, создается переменная `$nullDate`, для чего вызывается метод `getNullDate()` из класса `JDatabase`. Следует, однако, иметь в виду, что значение, присваиваемое этой переменной, на самом деле не является пустым. Например, в базе данных MySQL метод `getNullDate()` возвращает поле даты и времени со всеми нулями (**0000-00-00 00:00:00**). Но самое главное, что здесь используется то же самое значение даты и времени, что и при создании статьи.

Далее составляется первый запрос для получения сведений об авторе, необходимых для текущей статьи. И здесь нам придется принять еще одно проектное решение. В Joomla с каждой статьей связан столбец `created_by`, в котором находится идентификатор пользователя (`id`), являющегося автором статьи, а также столбец `created_by_alias` с дополнительным текстом, который может быть введен при создании или редактировании статьи. Какой же столбец следует выбрать, чтобы проверить, принадлежит ли очередная статья тому же самому автору?

Из этого положения можно выйти несколькими путями. В частности, можно выбрать дополнительный столбец `created_by_alias`, если таковой имеется в таблице базы данных, а иначе — столбец `created_by`. Еще проще произвести поиск на совпадение в любом из двух столбцов. А кроме того, можно предоставить пользователю возможность самому выбирать, как поступить дальше, с помощью отдельного параметра. В данном примере мы выберем второй, самый простой способ: составить список всех статей, совпадающих с любым из двух столбцов. Для этого нам нужно получить оба этих столбца для текущей статьи, чтобы воспользоваться ими во втором запросе.

Первый запрос составляется очень просто. Сначала создается новый объект запроса с помощью метода `getQuery()`. Логическое значение `true`, передаваемое этому методу в качестве аргумента, означает, что в ответ будет получен новый пустой запрос. Затем в запрос вводятся три столбца с помощью метода `select()`. К упомянутым выше двум столбцам добавлен еще и столбец `id`, хотя формально делать это совсем не обязательно. Но его добавление не требует никаких дополнительных затрат, а лишь упрощает отладку и проверку того, что по данному запросу возвращается ожидаемый результат.

В методе `from()` указывается таблица `#__content`. Напомним, что в запросах базы данных всегда используется префикс `#__`. Он заменяется конкретным префиксом таблицы перед тем, как передавать запрос базе данных. Далее в запрос вводится оператор `WHERE` с помощью метода `where()`, как показано ниже.

```
$query->where('id = ' . (int) $id);
```



В данном случае приведение типа переменной `$id` к целочисленному типу не требуется. Ведь для установки ее значения ранее был вызван метод `JRequest::getInt()`, а следовательно, ее значение и так является целочисленным. Тем не менее переменные, указываемые в операторе `WHERE` запросов SLQ, рекомендуется подвергать дополнительной обработке. Так, текст можно дополнительно обработать методом `quote()` из класса `JDatabase`, заключающим текст в кавычки и экранирующим специальные символы,

в том числе и сами кавычки, чтобы злоумышленник не смог подделать запрос. А целые значения переменных можно дополнительно обработать с помощью оператора приведения типов, как в данном примере. Подробнее о защите от умышленного внесения запросов SQL речь пойдет в главе 11.

В следующей строке кода вызывается метод `setQuery()` для объекта `$db`. Этот метод устанавливает поле в объекте базы данных, но на самом деле он не выполняет запрос. Это делается в строке

```
$currentArticle = $db->loadObject();
```

где в качестве значения возвращается объект.

Очень важно понять, что результаты запросов могут быть получены самыми разными методами. Для этой цели обычно применяются следующие методы.

- Метод `loadObject()`. Возвращает первую строку из запроса в виде объекта, где каждый столбец из строки является полем данного объекта.
- Метод `loadObjectList()`. Возвращает массив всех строк из запроса, где каждый элемент массива является объектом из данной строки.
- Метод `loadResult()`. Возвращает первый столбец из первой строки запроса.

Выбор конкретного метода зависит от типа запроса. В данном примере заранее известно, что в результате запроса будет возвращена максимум одна строка, поскольку выбирается строка, имеющая идентификатор (`id`) в качестве одного из значений. Каждый идентификатор в таблице `#__content` не похож на остальные, а следовательно, в результате запроса нельзя получить больше одной строки. Именно поэтому и вызывается метод `loadObject()`, возвращающий первую строку из запроса.

Если бы нам потребовался только первый столбец из первой строки, мы могли бы вызвать метод `loadResult()`. А если бы нам потребовалось несколько строк из запроса, мы бы, скорее всего, вызвали метод `loadObjectList()`. Имеются также методы для заполнения массивов результатами запроса вместо объектов. Такие методы можно обнаружить в классе `JDatabase` из файла `libraries/joomla/database/database.php`.

Итак, мы получили три столбца для текущей статьи в переменной `$currentArticle`. В листинге 6.10 приведена третья часть вспомогательного файла PHP.

Листинг 6.10. Исходный код из файла `helper.php`, часть 3

```
// запросить базу данных с целью получить статьи текущего автора
$query->clear();
$query->select('a.*');
$query->select('c.access AS cat_access, c.published AS cat_state, c.alias
↳AS cat_alias');
$query->from('#__content AS a');

// для ссылки на статьи и проверки прав доступа к ним
// требуются сведения о категории
$query->leftJoin('#__categories AS c ON c.id = a.catid');

// показывать только опубликованные статьи
$query->where('a.state = 1');
$query->where('c.published = 1');
```

```
// показывать только место доступа к статье и категории
$query->where('a.access IN (' . $levels . ')');
$query->where('c.access IN (' . $levels . ')');

// показывать только место, где статьи опубликованы в настоящий момент
$query->where('(a.publish_up = '.$db->Quote($nullDate).' OR
↳a.publish_up <= '.$db->Quote($now).')');
$query->where('(a.publish_down = '.$db->Quote($nullDate).' OR
↳a.publish_down >= '.$db->Quote($now).')');

// проверить автора статьи
if ($currentArticle->created_by_alias) {
    // если у текущей статьи имеется псевдоним автора, проверить
    // на совпадение столбец created_by или created_by_alias
    $query->where('(a.created_by =' . (int) $currentArticle->
↳created_by . ' OR ' . 'a.created_by_alias =
↳' . $db- >quote($currentArticle->created_by_alias) . ')');
} else {
    // если у текущей статьи отсутствует псевдоним автора, проверить
    // на совпадение только столбец created_by column
    $query->where('a.created_by =' . (int) $currentArticle->created_by);
}
```

В этой части исходного кода из вспомогательного файла PHP начинается формирование основного запроса. Сначала старый запрос очищается с помощью метода `clear()`, а затем составляются критерии выборки информации из базы данных. Синтаксическая конструкция `"a.*"` предписывает базе данных включить в запрос все столбцы из таблицы `"a"`. Для данного модуля из таблицы `#__content` требуется выбрать лишь несколько столбцов: название статьи, идентификатор и категорию. Не следует, однако, забывать, что всем пользователям разрешается составлять альтернативные компоновки для данного модуля, и поэтому заранее неизвестно, какие именно сведения о статьях предполагается отображать с помощью этих компоновок.

Базе данных не потребуется много времени, чтобы извлечь все нужные столбцы из таблицы `#__content`, что дает максимум удобств для составления всех последующих компоновок. В данный запрос включены также столбцы `access`, `published` и `alias` из таблицы `#__categories`. Они указываются в запросе с префиксом `"c."` этой таблицы, как показано ниже.

```
$query->select('c.access AS cat_access, c.published AS cat_state, c.alias
↳AS cat_alias');
```

В разных таблицах нередко используются одинаковые имена столбцов. Например, в обеих таблицах, `#__content` и `#__categories`, имеются столбцы `access` и `alias`. Результаты данного запроса будут возвращены в виде объекта, где имя столбца совпадает с именем поля. Следовательно, когда имена столбцов дублируются, их нужно изменить в запросе. И это делается с помощью псевдонима столбца `AS` таким же образом, как и для псевдонимов таблиц. Например, выражение `c.access AS cat_access` означает, что в результате запроса будет возвращено значение из столбца `access` для таблицы `#__categories` со столбцом `cat_access`. Подобным образом можно отличить столбец `access` для доступа к статье от столбца `cat_access` для доступа к категории.

В следующей строке кода таблица #__content обозначается как первая таблица в запросе с помощью псевдонима "a". Как правило, первой в запросе обозначается основная таблица, но в данном случае нас больше всего интересует информация из таблицы #__content.

Далее следует строка кода

```
$query->leftJoin('#__categories AS c ON c.id = a.catid');
```

где применяется соединение — способ ввода информации из другой таблицы в запрос. В данном случае три столбца из таблицы #__categories вводятся в каждую строку запроса. Как же тогда узнать, какую именно строку из таблицы #__categories следует использовать? На это указывает оператор ON. Выражение ON c.id = a.catid' предписывает базе данных найти строку в таблице #__categories, где столбец id совпадает со столбцом catid из таблицы #__content.

Для этой цели вызывается метод leftJoin(). Таблицы из базы данных могут быть соединены в запросе разными способами. Чаще всего используются левое и внутреннее соединения. Как правило, они делают одно и то же, а отличаются лишь тем, что именно происходит, если во второй таблице отсутствует строка, совпадающая со значением в операторе ON. При левом соединении в любых столбцах, выбираемых из второй таблицы, устанавливается пустое значение NULL. А при внутреннем соединении вся строка из первой таблицы исключается из запроса. Более подробно соединения рассматриваются в главе 11.

Для рассматриваемого здесь запроса это не имеет никакого значения. Если база данных не испорчена, то можно ожидать получение вполне достоверной категории для каждой статьи. Но если по какой-нибудь причине категория статьи удалена, эта статья все равно останется в запросе. А если бы нам понадобилось исключить из запроса статьи, для которых отсутствует категория, мы бы тогда вызвали метод innerJoin() вместо метода leftJoin().

Далее следует целый ряд вызовов метода where() для задания критериев выборки информации по данному запросу. В первых двух вызовах метода where() указывается требование, чтобы статья и ее категория были опубликованы. Следует заметить, что в таблице статей для отметки состояния "опубликовано" используется столбец state, тогда как в таблице категорий — столбец published. Единичное значение в обоих столбцах обозначает одно и то же: состояние "опубликовано".

А зачем вообще знать, опубликована ли категория? Дело в том, что в Joomla категории служат своего рода вместилищами для статей и других видов содержимого, подобно папкам для файлов на жестком диске компьютера. Если категория установлена в состояние "не опубликовано", значит, и все стати этой категории должны быть установлены в состояние "не опубликовано". Именно это правило и соблюдается в первых двух вызовах метода where().

Аналогичная логика применяется и к уровням доступа. Так, если доступ к категории ограничен, то ограничение накладывается и на все статьи данной категории. Напомним, что переменной $levels присвоен разделяемый запятыми список уровней доступа пользователя к статьям для их просмотра. В операторе IN запроса SQL проверяется наличие значения в разделяемом запятыми списке. Так, в операторе

```
a.access IN (1,2,3)
```

проверяется, равно ли **1**, **2** или **3** значение, извлекаемое из столбца `a.access` таблицы. Поэтому в рассматриваемом здесь коде в запрос вводятся два оператора `IN`: один — для доступа к статьям, другой — для доступа к категориям.

При построении символьной строки в методе `where()` следует проявлять особую аккуратность, употребляя открывающие и закрывающие круглые скобки. Вероятно, при дальнейшем усовершенствовании класса `JDatabaseQuery` в него будут добавлены методы, поддерживающие различные типы операторов сравнения, в том числе `IN`, `BETWEEN`, `LIKE`, **=** и т.д. И тогда можно будет воспользоваться выражением

```
$query->whereIn('a.access', $levels);
```

чтобы упростить операции сравнения типа `IN` и не вводить круглые скобки вручную. Но такие методы пока что отсутствуют. Впрочем, программное обеспечение постоянно совершенствуется, в особенности такой открытый проект, как Joomla. С другой стороны, можно и самостоятельно создать подкласс, производный от класса `JDatabaseQuery`, добавив в него нужные методы.

В следующем фрагменте кода проверяются даты начала и окончания публикации статьи (`publish_up` и `publish_down`), чтобы выяснить, опубликована ли статья на текущий момент. Для этой цели используются созданные ранее переменные `$now` и `$nullDate`. Необходимо проверить одно из двух:

- начальная или конечная дата не введена
  или,
- текущая дата находится в пределах между начальной и конечной датами.

Поэтому в данном фрагменте кода вводятся два оператора `WHERE`: один — для проверки начальной даты, другой — для проверки конечной даты, причем в каждом из них используется оператор `OR`. Обратите внимание на то, что все проверяемое значение заключается в круглые скобки. Например, результат вызова первого метода, реализующего оператор `WHERE`, может выглядеть следующим образом:

```
WHERE (a.published_up = '0000-00-00 00:00:00' OR a.publish_up
<= '2012- 12- 31 23:30:00')
```

Очень важно понять, что по умолчанию операторы `WHERE`, формируемые методом `where()`, связаны с оператором `AND` запроса SQL. Это означает, что *все* условия в операторах `WHERE` должны быть истинны для строки, извлекаемой по запросу. Но если поместить оба проверяемых условия в сигнатуре одного оператора `where()`, объединив их оператором `OR` и заключив в круглые скобки, то условие в операторах `WHERE` окажется истинным, если истинно любое из исходных условий, что, собственно, нам и требуется.

Кроме того, метод `$db->quote()` вызывается с целью заключить переменные `$nullDate` и `$now` в кавычки. И хотя заранее известно, что их значения действительны, кавычки все же нужны, поскольку эти значения могут содержать пробелы. Проверка конечной даты публикации организуется аналогичным образом. Но в данном случае проверяется, является ли текущая дата меньше конечной даты публикации.

В следующем фрагменте кода проверяется авторство статьи для текущего автора. Ранее было принято решение составить список всех статей, авторство которых совпадает со значениями в столбцах `created_by` или `created_by_alias`. Но что если в столбце

created_by_alias ничего не указано для текущей статьи. В таком случае проверка на совпадение со значением в столбце created_by_alias не должна производиться, поскольку нет никакого смысла показывать все остальные статьи с пустым значением в этом столбце. Поэтому сначала нужно проверить содержимое столбца created_by_alias для текущей статьи, а затем сформировать оператор WHERE в зависимости от того, что будет обнаружено в этом столбце. Если в столбце created_by_alias имеется какое-нибудь значение, оно используется в операторе WHERE следующим образом:

```
$query->where('(a.created_by =' . (int) $currentArticle->created_by
↳. ' OR ' . 'a.created_by_alias =
↳' . $db->quote($currentArticle->created_by_alias) . ')');
```

И в этом случае весь оператор WHERE заключается в круглые скобки.

Вторая часть условного оператора if выполняется при условии, что столбец created_by_alias пуст. В таком случае проверяется только значение в столбце created_by, как показано ниже.

```
$query->where('a.created_by =' . (int) $currentArticle->created_by);
```

Обратите внимание на то, что тип значения в столбце created_by специально приводится к целочисленному типу. Это значение и так должно быть целым, но во избежание программных ошибок значения, извлекаемые из столбцов таблицы, все-таки лучше специально приводить к нужному типу. А значение, извлекаемое из столбца created_by_alias, очень важно заключить в кавычки, вызвав метод $db->quote(). Ведь оно вводится пользователем в соответствующем поле при создании статьи, а следовательно, может оказаться каким угодно, в том числе и злонамеренным кодом. Если же заключить его в кавычки с помощью метода $db->quote(), то тем самым можно исключить внесение в запрос любого злонамеренного кода.

Четвертая, последняя часть вспомогательного файла PHP приведена в листинге 6.11.

Листинг 6.11. Исходный код из файла helper.php, часть 4

```
    // не показывать просматриваемую в настоящий момент статью в списке
    $query->where('a.id !=' . (int) $currentArticle->id);

    // если фильтр языков активизирован, показывать только те статьи,
    // которые написаны на текущем языке
    if ($app->getLanguageFilter()) {
        query->where('a.language IN (' . $db->
↳Quote(JFactory::getLanguage()->getTag()) . ',' . $db->Quote('*') . ')');
    }

    // задать порядок сортировки результатов запроса, исходя из
    // действительного параметра, значение которого определяется
    // в форме выражением validate="options"
    $query->order($params->get('article_ordering', 'a.title') . ' ' .
↳$params->get('article_ordering_direction', 'ASC'));
    // наложить на запрос ограничение с помощью параметра count
   // (примечание: разбиение на страницы в данном модуле не предполагается)
    $db->setQuery($query, 0, $params->get('count', 5));

    // получить список строк
```

```
    $items = $db->loadObjectList();

    // создать поле ссылки на каждый элемент списка, используя
    // класс маршрутизатора содержимого
    foreach ($items as &$item) {
        $item->slug = $item->id.':'.$item->alias;
        $item->catslug = $item->catid.':'.$item->cat_alias;
        $item->link = JRoute::_(ContentHelperRoute::getArticleRoute($item->
↳slug, $item->catslug));
    }
  }

  return $items;
}
```

В первой строке этой части исходного кода из вспомогательного файла PHP вводится условие с целью исключить текущую строку из списка. Ведь и так известно, что она написана текущим автором, и поэтому ее не следует включать в список искомых статей.

Следующий фрагмент кода тесно связан со средством Language Switcher (Переключатель языков), внедренным в версии Joomla 1.6. Это средство дает пользователю возможность помечать статьи, написанные на конкретном языке. Оператор `WHERE` вводится в запрос только в том случае, если данное средство активизировано, что проверяется при вызове метода `$app->getLanguageFilter()`. Если средство Language Switcher активизировано, то далее проверяется, написана ли статья на текущем языке (столбец `a.language`), или же язык особого значения не имеет, на что указывает критерий, обозначаемый знаком звездочки (`"*"`). Если же средство Language Switcher не активизировано, данный кодовый блок пропускается, а следовательно, оператор `WHERE` не вводится в запрос.

В следующем фрагменте кода параметры упорядочения используются для сортировки результатов запроса. В частности, метод `order()` вводит оператор `ORDER BY` в запрос. Напомним, что значения параметров рассматриваемого здесь модуля размечены для встраивания непосредственно в оператор `ORDER BY`, а для того чтобы обеспечить соответствие введенного значения одному из заданных вариантов выбора, служит выражение `validate="options"`. Благодаря этому значения параметров могут быть использованы непосредственно в запросе с помощью метода `order()`.

Следует заметить, что устанавливаемые по умолчанию значения предоставляются для каждого параметра. Как правило, эти значения достоверны, но иногда они могут таковыми и не оказаться. Допустим, создается новая версия рассматриваемого здесь модуля и в ней вводится еще один параметр. Если после создания старой версии данного модуля перейти к новой, этот новый параметр окажется неопределенным в новой версии модуля. И даже если указать действительное значение нового параметра, используемое по умолчанию, новая версия модуля будет все равно работать корректно, несмотря на то, что этот параметр фактически не определен.

И последнее действие перед выполнением запроса состоит в наложении на него ограничения с помощью параметра `count`. В большинстве запросов, составляемых для Joomla, используются параметры смещения и ограничения, указывающие базе данных MySQL начало и окончание обработки запроса. В данном случае начинать обработку запроса следует с самого начала (т.е. с нулевой строки таблицы), а завершать — номером строки,

определяемой параметром `count` (по умолчанию его значение равно **5**). Соответствующие аргументы передаются методу `setQuery()`.

В следующей строке кода выполняется запрос, а результаты его обработки возвращаются в виде массива объектов из рассмотренного ранее метода `loadObjectList()`. В итоге мы получаем в свое распоряжение массив объектов, каждый из которых содержит значения, извлеченные из одной строки таблицы в ходе обработки запроса. В составляемый список статей нам нужно включить ссылки на отдельные статьи. Поэтому нам остается лишь сформировать эти ссылки.

Сформировать ссылки в Joomla совсем не трудно, если воспользоваться для этой цели подходящими встроенными методами. А ссылки на статьи формируются по определенным правилам. В частности, для этой цели на веб-сайте обычно находится пункт меню, наиболее подходящий для статьи. Если для каждой статьи имеется отдельный пункт меню, то формируется ссылка, по которой этот пункт меню выбирается. В противном случае находится пункт меню для категории, которой принадлежит статья. Для каждой статьи мы попытаемся воспользоваться уже имеющимся URL вместо того, чтобы формировать новый URL. Благодаря этому ссылки на одну и ту же статью, уже имеющиеся в разных местах веб-сайта, будут приводить на одну и ту же его страницу по тому же самому URL. Как же все это сделать? К счастью, в Joomla уже имеются методы для формирования ссылок на статьи.

Итак, рассмотрим последний фрагмент кода, где в цикле `foreach` организуется поочередное обращение ко всем элементам массива `$items`. В этом цикле каждый объект из массива доступен в переменной `$item`. В двух первых строках кода в цикле `foreach` создаются поля для статьи и ее категории. В этих полях содержится идентификатор, знак двоеточия (`:`) и псевдоним (например, `23:my-article-alias`). Все это потребуется при создании URL для ссылки на статью.

Ниже приведена третья строка кода из цикла `foreach`.

```
$item->link = JRoute::_(ContentHelperRoute::getArticleRoute($item->slug,
↳$item->catslug));
```

Именно в этой строке кода и формируется ссылка на отдельную статью. И делается это в две стадии. Сначала с помощью метода `getArticleRoute()` из класса `ContentHelperRoute` составляется маршрутизация статьи. В качестве аргументов этот метод принимает заготовки статьи и ее категории, полученные в двух предыдущих строках кода, и возвращает запрос на формирование URL. Как упоминалось ранее в этой главе, каждый модуль обычно связан с некоторым компонентом. Так, рассматриваемый здесь модуль связан с компонентом `com_content`. Поэтому формируемые в нем ссылки на статьи должны быть согласованы со ссылками, формируемыми в самом компоненте `com_content` или в других модулях, использующих этот компонент.

При создании компонента имеется, в частности, возможность установить определенные правила для маршрутизации. В компоненте `com_content` это делается с помощью метода `getArticleRoute()`, находящего имеющийся пункт меню как наиболее подходящий для данной статьи. Так, если имеется пункт меню для отдельной статьи, то будет создан указывающий на него URL. В противном случае находится пункт меню, указывающий на категорию данной статьи. И это делается потому, что ссылки на отдельную статью должны, как правило, направлять по тому же самому URL.

Самое главное, чтобы ссылки на статьи были такими же, как и те, что сформированы в компоненте или в других модулях статей. И для этого мы должны воспользоваться теми же самыми методами формирования ссылок. Этим гарантируется, что получаемые в итоге ссылки будут совместимы с другими ссылками, а кроме того, экономится наш труд. Более того, если изменяется порядок маршрутизации статей или исправляется программная ошибка, все эти изменения будут подхвачены рассматриваемым здесь модулем.

Результат выполнения метода `getArticleRoute()` выглядит довольно просто, как в показанном ниже примере.

```
index.php?option=com_content&view=article&id=1:administrator
↳components&catid=21:components&Itemid=273
```

Первая часть ссылки на статью остается неизменной: `index.php?option=com_content&view=article`. А две следующие части являются созданными ранее заготовками статьи и категории. Но все эти части — не такое уж и большое достижение со стороны метода `getArticleRoute()`. Совсем другое дело — последняя часть `&Itemid=273`, которая означает пункт меню, наиболее подходящий для данной статьи по правилам, установленным для компонента. Именно такой пункт меню и возвращает метод `getArticleRoute()`.

Получив результаты, возвращаемые методом `getArticleRoute()`, мы должны превратить их в полноценный URL. И это делается с помощью метода `JRoute::_()`, который, в свою очередь, вызывает функцию маршрутизации, если таковая имеется и предоставляется компонентом. В данном случае это функция `ContentBuildRoute()` из файла `components/com_content/router.php`. Если установлена кнопка-переключатель **Yes** параметра **Search Engine Friendly URLs** на вкладке **SEO** диалогового окна **Global Configuration**, то URL преобразуется в удобный для поиска вариант, а запрос заменяется псевдонимами из пункта меню, а иногда — из категории. Так, приведенный выше пример ссылки преобразуется в следующий удобный для поиска URL:

```
<маршрутный путь Joomla!>/index.php/using-joomla/extensions/components/
↳administrator-components
```

Этот URL затем преобразуется в полноценный URL системным подключаемым модулем SEF, добавляющим адрес хоста (т.е. веб-узла, например, `http://www.mydomain`) в начале URL.

После обработки каждого объекта из массива `$items` метод `getList()` завершается. Сначала происходит выход из цикла `foreach`, затем из кодового блока условных операторов `if/then` и далее возврат массива `$items`.

Файл исходной компоновки

Следующим по порядку обработки следует файл исходной компоновки `default.php`. Если для модуля не создана и не указана альтернативная компоновка, именно этот файл возвращается при вызове метода `JModuleHelper::getLayoutPath()` в главном файле PHP данного модуля. Исходный код из файла исходной компоновки приведен в листинге 6.12.

Листинг 6.12. Исходный код из файла `default.php` исходной компоновки

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry.
```

```
 * All rights reserved.
 * @license      GNU General Public License version 2 or later;
 * see LICENSE.txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

// вывести список ссылок в элементе разметки ul
?>
<ul class="joompro<?php echo $moduleclass_sfx; ?>">
<?php foreach ($list as $item) : ?>
    <li>
        <a href="<?php echo $item->link; ?>">
            <?php echo $item->title; ?></a>
    </li>
<?php endforeach; ?>
</ul>
```

В этой простой компоновке сначала создается элемент `ul` разметки неупорядоченного списка в коде HTML, а затем выполняется цикл `foreach` для поочередного обращения к элементам массива `$item`. Напомним, что переменной `$item` присваивается массив, возвращаемый из вспомогательного метода в следующей строке кода из файла `mod_joompro_articles_author.php`:

```
$list = modJoomProArticlesAuthorHelper::getList($params);
```

Переменные, оказывающиеся в области действия там, где данный файл компоновки включается по команде `require`, по-прежнему находятся в области действия компоновки. В данной компоновке просто выполняются команды языка PHP, но не объявляется класс или функция, и поэтому она выполняется сразу, как если бы эта компоновка была составной частью программы, в которую она включена.

В цикле `foreach` создается элемент `li` разметки списка в коде HTML, в который включается ссылка на статью и ее название. По завершении цикла `foreach` файл компоновки закрывается после закрывающего дескриптора `ul`. Как и прежде, в рассматриваемой компоновке используется альтернативный синтаксис языка PHP, позволяющий без особого труда сочетать дескрипторы HTML с командами PHP.

Файлы языковой поддержки

Для рассматриваемого здесь модуля предусмотрены два файла языковой поддержки. В листинге 6.13 приведено содержимое системного файла языковой поддержки с расширением `.sys.ini`.

Листинг 6.13. Содержимое файла `en-GB.mod_joompro_articles_author.sys.ini`

```
; Системный файл языковой поддержки модуля mod_joompro_articles_author
; Файлы с расширением .sys.ini files используются при составлении списков
; расширений в диспетчере расширений или диспетчере подключаемых модулей

MOD_JOOMPRO_ARTICLES_AUTHOR="Articles by Current Author (sys file)"
MOD_JOOMPRO_ARTICLES_AUTHOR_DESCRIPTION="This module shows articles by the
↳same author as the article in view. (sys file)"
```

Как упоминалось ранее, этот файл служит для перевода названия и описания модуля при его отображении в списке, но не при открытии для редактирования. Характерными тому примерами служат установка модуля и его перечисление в компоненте Module Manager. Текст "(sys file)" введен в этот файл ради наглядности, чтобы показать, какой именно файл языковой поддержки используется (в данном случае — системный).

В листинге 6.14 приведено содержимое еще одного файла языковой поддержки с расширением .ini.

Листинг 6.14. Содержимое файла `en-GB.mod_joompro_articles_author.ini`

```
; Основной файл языковой поддержки модуля mod_joompro_articles_author

MOD_JOOMPRO_ARTICLES_AUTHOR="Articles by Current Author"
MOD_JOOMPRO_ARTICLES_AUTHOR_DESCRIPTION="This module shows articles by the
↳same author as the article in view. (ini file)"
MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_LABEL="Number of Articles to Show"
MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_DESC="Enter the maximum number of
↳articles to show in the module."
MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_ARTICLEORDERING_LABEL="Article Ordering"
MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_ARTICLEORDERING_DESC="Select the column
↳that will be used to order the articles."
MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_HITS_VALUE="Hits"
MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_CREATED_VALUE="Date Created"
MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_STARTPUBLISHING_VALUE="Date Published"
MOD_ARTICLES_CATEGORY_FIELD_ARTICLEORDERINGDIR_LABEL="Ordering Direction"
MOD_ARTICLES_CATEGORY_FIELD_ARTICLEORDERINGDIR_DESC="Enter Low to High for
↳normal alphabetical sort. Enter High to Low to order by most recent or
↳most hits."
MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_DESCENDING_VALUE="High to Low"
MOD_JOOMPRO_ARTICLES_AUTHOR_OPTION_ASCENDING_VALUE="Low to High"
```

И в этом файле указывается название и описания модуля. Соответствующие ключи служат для отображения названия и описания модуля при редактировании его экземпляра. Текст "(ini file)" введен в этот файл ради наглядности, чтобы показать, какой именно файл языковой поддержки используется (в данном случае — основной). Все ключи из этого файла взяты из XML-файла данного модуля, поскольку в результатах, выводимых из данного модуля, текст вообще отсутствует. А если бы текст присутствовал, то для него в этот файл были бы включены соответствующие языковые ключи.

Проверка достоверности параметров средствами класса `JForm`

Класс `JForm` внедрен в версии Joomla 1.6 и предоставляет отличные средства для проверки достоверности параметров, вводимых пользователями. Покажем, как это делается, на двух наглядных примерах.

Проверка значений во вспомогательном методе

Вернемся к полю `count` из XML-файла, где указывается максимальное количество статей, отображаемых в данном модуле. В настоящий момент имеется следующий код:

```
<field
    name="count"
    type="text"
    default="5"
    label="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_LABEL"
    description="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_DESC">
</field>
```

Значение должно быть положительным целым, чтобы считаться достоверным. Но в данный момент код позволяет ввести в текстовом виде не только положительное, но и отрицательное целое и нулевое значение. А это приведет к тому, что запрос будет выполняться без каких-либо ограничений. В данном случае будут перечислены все статьи указанного автора. На крупном веб-сайте таких статей может оказаться десятки и даже сотни, что, вероятно, затруднит их отображение, а возможно, даже скажется на производительности веб-сайта.

Во избежание подобных осложнений мы можем усовершенствовать рассматриваемый здесь код несколькими способами. В частности, организовать проверку значения во вспомогательном методе, изменяя его на используемое по умолчанию, если оно оказывается вне заданных пределов. Для этого можно, например, ввести следующий фрагмент кода:

```
$count = (int) $params->get('count', 5);
$count = ($count <= 0 || $count > 10) ? 5 : $count;
$db->setQuery($query, 0, $count);
```

Благодаря такой проверке достоверности мы можем быть всегда уверены в том, что введенное значение находится в пределах от **1** до **10**.

Применение целочисленного типа в классе `JForm`

Недостаток рассмотренного выше способа заключается в том, что мы не препятствуем пользователю ввести недействительное значение и никак не предупреждаем его об этом. Используя класс `JForm`, мы можем далее усовершенствовать проверку достоверности во время ввода данных самыми разными средствами. В частности, воспользоваться целочисленным типом данных, чтобы ограничить возможности их ввода списком целых значений. Ниже приведен пример того, как можно изменить определение поля в XML-файле.

```
<field
  name="count"
  type="integer"
  first="1"
  last="10"
  step="1"
  default="5"
  label="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_LABEL"
  description="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_DESC">
</field>
```

В этом фрагменте кода тип вводимых данных был изменен с текстового (`"text"`) на целочисленный (`"integer"`). При загрузке формы редактирования модуля поле ввода данных будет представлено в раскрывающемся списке, как показано на рис. 6.2.

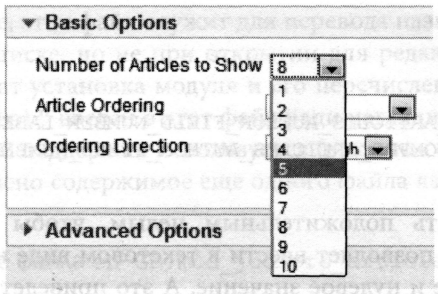


Рис. 6.2. Пример поля ввода целочисленных данных

Преимущество данного способа заключается в том, что пользователь может ввести лишь достоверные данные, но только в предварительно заданных пределах. И хотя в данном примере такое ограничение не является существенным, тем не менее возможны ситуации, где действительными оказываются числовые значения в пределах от **1** до **1000**, а выводить все эти значения из списка вряд ли удобно.

В качестве выхода из этого затруднительного положения можно воспользоваться атрибутом `step`, чтобы выводить значения с *б*ольшим приращением, например `50` или `100`, и тем самым уменьшить количество вариантов выбора из списка. И в некоторых случаях это может действительно стать удобным выходом из положения.

Еще один недостаток смены типа данных на целочисленный в атрибуте `type="integer"` заключается в том, что злоумышленнику ничто не мешает ввести недействительное число, используя такое инструментальное средство, как, например, Firebug. Безусловно, данный способ упрощает ввод данных пользователем, но не защищает от злоумышленников.

Фильтрация целых значений средствами класса `JForm`

Еще один способ состоит в том, чтобы воспользоваться атрибутом `filter`. Для фильтрации данных, вводимых в форме при сохранении модуля, используются фильтры, предварительно заданные в Joomla. Так, если изменить тип данных обратно на текстовый и добавить атрибут `filter`, рассматриваемый здесь код примет следующий вид:

```
<field
  name="count"
  type="text"
  filter="integer"
  default="5"
  label="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_LABEL"
  description="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_DESC">
</field>
```

Такой способ позволяет автоматически преобразовать любое нецелочисленное значение, введенное в поле, в нулевое. В данном примере это не особенно помогает, поскольку мы получаем такой результат и без всякой фильтрации. Но о таком способе все же стоит знать.



Одним из преимуществ атрибута `filter` является его безопасность. Данные, вводимые в окне браузера, не могут быть подделаны манипуляциями с HTML-формой, поскольку их фильтрация происходит в процессе сохранения. Фильтрации подлежат все

стандартные типы данных, поддерживаемые в классе JFilterInput, в том числе целочисленный (integer), с плавающей точкой одинарной и двойной точности (float и double), логический (boolean), слово (word), командный (cmd), по основанию 64 (base64), строковый (string), гипертекстовый (html), массив (array), путь (path), а также имя пользователя (username). Кроме того, новые типы фильтров добавлены в класс JForm, в том числе rules, unset, raw, int_array, safehtml, server_utc, а также user_utc. Более подробно доступные типы фильтров рассматриваются в приложении Б.

Применение специального класса JFormRule

Еще один способ состоит в создании специального правила для поля. Правила проверки достоверности доступны с помощью атрибута validate. В Joomla установлен целый ряд правил проверки достоверности вводимых данных, включая логические, электронную почту, равенство, варианты выбора, телефон, URL, а также имя пользователя. Этим правилам соответствуют классы из папки libraries/joomla/form/rules. Если ни одно из этих предварительно установленных правил не подходит для проверки достоверности вводимых данных, можно установить свои собственные специальные правила именно с таким кодом, какой требуется. Для этого в XML-файл необходимо внести следующие изменения:

```
<fieldset name="basic" addrulepath="modules/mod_joompro_articles_author">
<field
    name="count"
    type="text"
    validate="countinteger"
    filter="integer"
    default="5"
    label="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_LABEL"
    description="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_DESC">
</field>
```

Сначала в элемент разметки fieldset вводится атрибут addrulepath с указанием пути к папке со специальным правилом. В данном случае она размещается на самом верхнем уровне структуры папок рассматриваемого здесь модуля. Затем в атрибуте validate элемента разметки field указывается имя файла с правилом (в данном случае countinteger). Далее атрибут filter="integer" добавляется с целью вынудить систему сохранить введенное значение как целое. Так, если кто-нибудь введет значение **15,5**, оно должно быть сохранено как **15**.

После этого в папку рассматриваемого здесь модуля вводится файл countinteger.php, в котором объявляется класс, реализующий специальное правило проверки вводимых данных на соответствие целым значениям. Ниже приведен исходный код из этого файла.

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter & Louis Landry. All rights
↳reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

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

```
jimport('joomla.form.formrule');

class JFormRuleCountInteger extends JFormRule
{
  public function test(& $element, $value, $group = null, & $input = null,
↳& $form = null)
  {
      return ((int) $value > 0 && (int) $value <= 30);
  }
}
```

Этот класс называется `JFormRuleCountInteger` и расширяет класс `JFormRule`. В нем определен только один метод `test()`, поскольку другие методы не требуются. В качестве одного из его аргументов указывается переменная `$value`, которой присваивается значение, введенное в форме. В данном примере проверяется, находится ли введенное значение в пределах от **1** до **30**. В противном случае будет выдано сообщение об ошибке `"Invalid field: Number of Articles to Show"` (Недействительное поле: количество отображаемых статей), а форма не будет сохранена.

Как видите, минимально и максимально допустимые значения в проверяемом поле жестко запрограммированы величинами **1** и **30**. Логику работы класса `JFormRuleCountInteger` можно сделать намного более развитой и гибкой, если устанавливать минимально и максимально допустимые значения в элементе `field` разметки формы из XML-файла. Для этого в элемент разметки `field` из данного XML-файла вводятся атрибуты `minimum` и `maximum`, как выделено полужирным в приведенном ниже коде.

```
<field
    name="count"
    type="text"
    validate="countinteger"
    minimum="1"
    maximum="10"
    filter="integer"
    default="5"
    label="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_LABEL"
    description="MOD_JOOMPRO_ARTICLES_AUTHOR_FIELD_NUMBER_DESC">
</field>
```

Далее необходимо внести изменения в метод `test()` из класса `JFormRuleCountInteger`, чтобы воспользоваться новыми атрибутами, как показано ниже.

```
public function test(& $element, $value, $group = null, &
↳$input = null, & $form = null)
  {
    $max = (int) $element->getAttribute('maximum') ?
↳$element->getAttribute('maximum') : 30;
    $min = (int) $element->getAttribute('minimum') ?
↳$element->getAttribute('minimum') : 1;
    return ((int) $value >= $min && (int) $value <= $max);
}
```

В приведенном выше фрагменте кода тернарный оператор языка PHP применяется для установки максимально и минимально допустимых значений в переменных `$max` и `$min`

соответственно. Данный метод должен оказаться работоспособным даже в том случае, если атрибуты maximum и minimum не установлены правильно. Для этого проверяется, установлено ли в них положительное целое значение. И если такое значение установлено, то оно используется далее, а иначе — выбираются устанавливаемые по умолчанию значения **30** и **1** соответственно.

В данном коде можно также изменить пределы допустимых значений, установив соответственно атрибуты maximum и minimum в XML-файле. Таким образом, установленное нами специальное правило оказывается намного более гибким и полезным не только в рассматриваемом здесь случае, но и в ряде других ситуаций.

Сообщение об ошибке проверки достоверности

Как было показано ранее, когда пользователь вводит недействительное значение параметра count, выводится сообщение об ошибке "Invalid field: Number of Articles to Show". В нем пользователь уведомляется о том, что в поле ввода данных что-то не так, но не поясняется, что именно. Это сообщение об ошибке выводится по умолчанию, если не определить явно другое сообщение. Но это другое сообщение можно определить двумя способами.

Первый способ состоит в том, чтобы добавить атрибут message в элемент разметки field. Например, в элемент field разметки поля параметра count можно было бы ввести следующую строку кода:

```
message="MOD_JOOMPRO_ARTICLES_AUTHOR_COUNTINTEGER_MESSAGE"
```

А затем добавить приведенную ниже строку в основной файл языковой поддержки с расширением .ini.

```
MOD_JOOMPRO_ARTICLES_AUTHOR_COUNTINTEGER_MESSAGE="Article count must be
↳between 1 and 10."
```

Если теперь попытаться ввести значение вне пределов, заданных для параметра count, то из основного файла языковой поддержки будет извлечено и выведено на экран сообщение об ошибке "Article count must be between 1 and 10" (Количество статей должно быть в пределах от 1 до 10).

Другой способ состоит в том, чтобы возвратить объект типа Jexception с сообщением из метода test() установленного правила проверки достоверности, если введенное значение окажется недействительным. Этого можно добиться следующим образом: прежде всего удалить атрибут message из элемента разметки field в XML-файле, а затем внести в метод test() следующие изменения:

```
  public function test(& $element, $value, $group = null, & $input =
↳null, & $form = null)
  {
    $max = (int) $element->getAttribute('maximum') ?
↳$element->getAttribute('maximum') : 30;
    $min = (int) $element->getAttribute('minimum') ?
↳$element->getAttribute('minimum') : 1;
    $result = ((int) $value >= $min && (int) $value <= $max);

    // построить объект типа JException
    if ($result === false) {
        $result = new JException(JText::sprintf(
```

```
↳'MOD_JOOMPRO_ARTICLES_AUTHOR_COUNTINTEGER_MESSAGE', $min, $max));
    }

    return $result;
```

Переменные `$max` и `$min` устанавливаются, как и прежде. Затем переменной `$result` присваивается логическое значение `true` или `false`. Если это логическое значение `false`, то в переменной `$result` сохраняется новый объект типа `JException` с сообщением об ошибке. Следует заметить, что для вывода этого сообщения вызывается метод `JText::sprintf()`, что дает возможность отображать значения переменных `$max` и `$min` в самом сообщении.

Затем возвращается значение переменной `$result`, которое будет логическим значением `true`, если данные в поле введены правильно, а иначе — объектом типа `JException` с сообщением об ошибке. И наконец, в основной файл языковой поддержки с расширением `.ini` необходимо добавить следующую строку:

```
MOD_JOOMPRO_ARTICLES_AUTHOR_COUNTINTEGER_MESSAGE="Article count must be
↳between %1$d and %2$d."
```

В этой строке метки-заполнители `%1$d` и `%2$d` служат для указания мест вывода значений переменных `$max` и `$min`. Они будут вставлены в сообщение об ошибке с помощью метода `JText::sprintf()`. В данном случае сообщение об ошибке, выводимое для пользователя, будет таким же, как и в предыдущем примере. Но если изменить значения атрибута `minimum` или `maximum` в XML-файле, то эти изменения автоматически появятся и в сообщении об ошибке.

Проверка достоверности списка статей

Напомним, что ранее в XML-файл был введен атрибут `validate="options"` для установки параметров `article_ordering` и `article_ordering_direction`. Это было сделано с целью не дать злоумышленнику возможность ввести в форму злонамеренный код SQL в обход проверки достоверности. Посмотрим, как эта проверка действует на практике.

Как и в предыдущем примере, поле объекта типа `JForm` размечается атрибутом `validate`, а при сохранении формы в теле метода `validate()` из класса `JForm` происходит обращение к классу `JFormRule`, в котором установлено правило проверки достоверности вводимых данных. Значение данного атрибута должно совпадать с именем файла, в котором объявлен класс `JFormRule`. В данном случае атрибут `validate` имеет значение `"options"`, а следовательно, файл должен иметь имя `options.php`, тогда как класс — `JFormRuleOptions`. Как упоминалось ранее, классы основных правил проверки достоверности хранятся в папке `libraries/joomla/form/rules`. Но поскольку данное правило является основным, то указывать путь к упомянутой папке в атрибуте `addrulepath` из XML-файла совсем не обязательно.

В листинге 6.15 приведен исходный код из файла `options.php`.

Листинг 6.15. Исходный код из файла `libraries/joomla/form/rule/options.php`

```
<?php
/**
 * @package Joomla.Platform
```

```
 * @subpackage Form
 * @copyright     Copyright (C) 2005 - 2012 Open Source Matters, Inc. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

defined('_JEXEC') or die;

jimport('joomla.form.formrule');
/**
 * Класс правила проверки достоверности формы для платформы Joomla!.
 * Требует, чтобы вводимое значение было одним из вариантов выбора в
 * поле type="list"
 *
 * @package       Joomla.Platform
 * @subpackage    Form
 * @since 11.1
 */
class JFormRuleOptions extends JFormRule
{
    /**
     * Метод для проверки достоверности значения.
     *
     * @param     object      $element    Объект типа JXMLElemen,
     *                                    представляющий дескриптор <field/>
     *                                    для разметки поля в форме.
     *
     * @param     mixed $value            Проверяемое значение в поле формы.
     * @param     string      $group      Значение, определяющее групповое
     *                                    имя поля. Служит в качестве массива,
     *                                    содержащего поле. Так, если имя поля
     *                                    определяется атрибутом name="foo", а
     *                                    групповое значение равно "bar", то
     *                                    поле будет иметь полное имя "bar[foo]".
     *
     * @param     object      $input      Дополнительный объект JRegistry
     *                                    с полным набором данных для проверки
     *                                    достоверности во всей форме в целом.
     *
     * @param     object$form             Объект формы, в которой поле проверяется
     *                                    на достоверность введенных данных.
     *
     * @return    boolean                 Истинно, если значение достоверно,
     *                                    а иначе — ложно.
     *
     * @since     11.1
     * @throws    JException              Исключение, генерируемое, если
     *                                    правило недействительно.
     */
    public function test(& $element, $value, $group = null, & $input =
↳null, & $form = null)
    {
        // проверить каждое значение и возвратить логическое значение
```

```
        // true, если введенное значение совпадает с допустимым.
        foreach ($element->option as $option) {
            if ($value == $option->getAttribute('value')) {
                return true;
            }
        }
        return false;
    }
}
```

Приведенный выше исходный код из файла `options.php` довольно прост. Как и прежде, объявляемый класс `JFormRuleOptions` расширяет класс `JFormRule` и имеет единственный метод `test()`. Переменная `$element` содержит элемент `field`, тогда как переменная `$value` — введенное пользователем значение.

При вычислении выражения `$element->option` в теле метода `test()` возвращается массив объектов типа `JXMLElement` — по одному на каждый элемент `option` из элемента разметки `field`. Далее введенное пользователем значение (переменной `$value`) проверяется в цикле на совпадение с одним из всех имеющихся вариантов выбора (значением атрибута `value` в элементе разметки `option`). И как только совпадение будет обнаружено, произойдет немедленный возврат логического значения `true` из цикла. Если же введенное значение не совпадет ни с одним из значений в массиве, определяющих имеющиеся варианты выбора, из метода `test()` будет возвращено логическое значение `false`.

Для опробования рассматриваемой здесь проверки достоверности измените форму (в частности, средствами Firebug), введя в ней недействительное значение параметра упорядочения статей (`article_ordering`; например, значение `"xxx"` вместо значения `"a.title"` атрибута `default`). При последующей попытке сохранить форму вы должны получить сообщение об ошибке `"Invalid Field: Article Ordering"` (Недействительное поле: упорядочение статей). А если вы удалите атрибут `validate="options"` из XML-файла данного модуля и сделаете еще одну попытку, форма будет сохранена.

Как видите, класс `JForm` представляет немало возможностей для точной настройки проверки достоверности данных, вводимых в полях формы. В частности, для ограничения возможных вариантов ввода данных можно выбрать соответствующие типы полей, для исправления недействительных данных в процессе их сохранения — воспользоваться фильтрами, а для того, чтобы воспрепятствовать пользователю сохранить недействительные данные, — установить правила проверки их достоверности. В последнем случае можно воспользоваться стандартными правилами или же установить свои собственные. Все эти способы можно сочетать, согласовывать и адаптировать под конкретные нужды, чтобы добиться желаемого режима работы заполняемой формы.

Справочный файл

Если предполагается распространение модуля, его придется снабдить файлом экранной справки. С этой целью можно, например, ввести URL справки в XML-файл, как показано ниже.

```
<help url="HELP_EXTENSIONS_MODULE_JOOMPRO_ARTICLES_AUTHOR_URL"/>
```

Элемент разметки `help` должен быть вложен в элемент разметки `extension`. Его принято располагать после элементов разметки `files` и `languages`. Но в данном примере

элемент разметки languages отсутствует, и поэтому элемент разметки help должен следовать сразу же после элемента разметки files.

В основной файл языковой поддержки с расширением .ini добавляется строка с языковым ключом для перевода справочной информации по указанному URL. Так, если имеется файл mod_joompro_articles_author_help.html с экранной справкой по рассматриваемому здесь модулю, то в основной файл его языковой поддержки с расширением .ini должна быть введена следующая строка:

```
HELP_EXTENSIONS_MODULE_JOOMPRO_ARTICLES_AUTHOR_URL=
↳"http://joomlaprogrammingbook.com/mod_joompro_articles_author_help.html"
```

Если теперь пользователь щелкнет кнопкой мыши на пиктограмме Help (Справка) непосредственно на панели инструментов при редактировании данного модуля, в окно будет загружена справочная информация из файла по назначенному URL. Размещение URL в файле языковой поддержки позволяет выводить на экран справочную информацию на избранном пользователем языке.

Упаковка модуля

Итак, мы создали вполне работоспособный модуль. А теперь его нужно сделать доступным на платформе Joomla. Это делается просто и таким же образом, как и для подключаемых в качестве расширения модулей. Для упаковки модуля в архивный файл выполните следующие действия.

1. Обновите XML-файл (mod_joompro_articles_author.xml) данного модуля, чтобы включить в него введенный ранее файл countinteger.php, добавив следующий элемент разметки filename в элемент разметки files:
   ```
   <filename>countinteger.php</filename>
   ```
2. Скопируйте содержимое папки modules/mod_joompro_articles_author во временную папку на жестком диске своего компьютера.
3. Воспользуйтесь программой архивирования, чтобы создать архив формата zip из файлов, находящихся во временной папке. Содержимое архива должно быть таким же, как и содержимое папки modules/mod_joompro_articles_author. Файл архива принято называть следующим образом: mod_joompro_articles_author_1.0.0.zip. Последние цифры обозначают номер версии модуля, принимая во внимание возможные его обновления впоследствии.

Проверьте, насколько успешно данный модуль устанавливается из полученного в итоге архивного файла. С этой целью удалите сначала существующую копию модуля, а затем установите новую его копию из архивного файла.

Краткий обзор созданного модуля

Подведем краткий итог, чего нам удалось добиться, создав новый модуль.

- Этот модуль органично вписывается в остальную часть веб-сайта на Joomla. Для пользователя он оказывается неотъемлемой частью веб-сайта на Joomla.

- В нем полностью поддерживаются многие языки. Мы можем создать два файла поддержки модуля на любом языке для отображения его текстовых надписей на избранном языке.
- В нем полностью поддерживаются альтернативные компоновки и замещения компоновки шаблонов. Вызывая метод `getLayoutPath()` для получения имени файла компоновки, мы предоставляем модулю столь эффективное средство.
- Он дает пользователю возможность вводить значения параметров или выбирать их варианты для управления режимом работы модуля.
- Он защищает от умышленного внесения запросов SQL и других возможных атак злоумышленников на веб-сайт. Используя типы полей, фильтрацию и проверку достоверности, мы осуществляем строгий контроль данных, вводимых пользователем, исключая возможность внедрения злонамеренных данных.
- Он предоставляет экранную справку.
- Он может быть установлен и удален на любом веб-сайте, построенном на Joomla 1.6 или более поздней версии.

Такой перечень функциональных возможностей созданного нами модуля впечатляет! А ведь мы реализовали их, просто воспользовавшись доступными средствами Joomla, создав модуль в соответствии со стандартными приемами разработки в Joomla.

Передовой опыт разработки и применения модулей

Ниже приведены некоторые рекомендации относительно разработки и применения модулей.

- Если при выполнении модуля возникает ошибка в базе данных, ее код не должен возвращаться из модуля. Как правило, модуль не является главным содержимым страницы, и поэтому нет никакой нужды останавливать обработку данных или пугать пользователя сообщениями об ошибках, к которым он не имеет никакого отношения. Таким образом, ошибка должна быть обработана внутри модуля, не вмешивая самого пользователя.
- Модули должны быть компактными и относительно быстро выполняемыми (т.е. легковесными). Не следует забывать, что модуль является одним из многих элементов содержимого, обрабатываемых на данной странице.

Резюме

В этой главе мы рассмотрели сначала основной модуль, а затем создали свой собственный. По ходу изложения материала было показано, каким образом модули могут быть связаны с компонентами и как организуется взаимодействие разнотипных файлов в модуле. Кроме того, были продемонстрированы различные способы возможной проверки достоверности и фильтрации данных, вводимых в формах.

При создании собственного модуля мы выгодно воспользовались многими средствами, доступными на платформе Joomla. И в конечном счете у нас получился модуль, органично вписывающийся в остальную часть веб-сайта на Joomla.

7

Компоненты, часть I. Контроллеры и модели

Компоненты обычно оказываются самыми крупными и сложными расширениями, и поэтому для их создания требуется как можно больше знаний, опыта и настойчивости. И если вы дочитали книгу до этой главы, значит, готовы принять столь трудный вызов.

В начале этой главы подробно рассматривается шаблон проектирования MVC (модель–контроллер–представление), применяемый при разработке всех основных компонентов в Joomla. И это будет сделано на примере административной части компонента Weblinks (Веб-ссылки). Сначала будет сделан краткий обзор файлов этого компонента, а затем подробно рассмотрены различные его функции. После этого будут исследованы функции, реализованные в классах контроллера и модели, а также рассмотрен класс `JTable` и его применение для взаимодействия с базой данных.

Что такое компонент

Типичный веб-сайт на Joomla состоит из двух стандартных блоков. Одним из них является вводимое или отображаемое содержимое, в том числе статьи, контактная информация, продукция и прочее, а другим — ряд пунктов меню, позволяющих перемещаться по веб-сайту и пользоваться его содержимым. Оба этих стандартных блока построены на компонентах. В административной части веб-сайта на Joomla находятся административные компоненты, предназначенные для управления всем его содержимым. А в пользовательской части веб-сайта каждый пункт меню основывается на компоненте.

Для выполнения любой задачи, кроме простейшего отображения информации, как правило, используется компонент. Так, если в базу данных вводится новая информация, например статья, для этой цели используется отдельный компонент. А когда выбирается пункт меню, то на самом деле осуществляется доступ к компоненту.

Как пояснялось в главе 3, каждый цикл обработки данных в Joomla начинается с выполнения файла, запускающего компонент (например, `components/com_content/content.php`). Для базовых компонентов в этом файле вызываются программы, составляющие структуру MVC данного компонента.

Одно из возможных недоразумений в отношении компонентов в Joomla связано с тем, что они применяются в административной части для создания веб-сайта и управления им. В частности, компоненты административной части веб-сайта служат для управления другими компонентами как в пользовательской, так и административной части веб-сайта, а те, в свою очередь, — для управления всеми остальными расширениями: модулями, шаблонами, подключаемыми модулями и средствами языковой поддержки. Например, диспетчер модулей является компонентом административной части, но его основное назначение — помочь в создании модулей и управлении ими на веб-сайте.

Основные функции компонентов

Назначение компонентов легче понять, если рассматривать их с точки зрения тех задач, которые поручаются Joomla. Так, если требуется только отобразить информацию, это можно сделать в модуле или подключаемом модуле. Но если потребуется программа для выполнения какого-нибудь действия, например, сохранения информации в базе данных, это, как правило, делается в компоненте. Иными словами, всякий раз, когда мы нажимаем кнопку передачи или сохранения данных в Joomla, сделанный нами запрос обрабатывается соответствующим компонентом.

В большинстве HTML-форм используется атрибут `method="post"` для обработки содержимого формы. И это делается практически во всех случаях, когда требуется программа для выполнения какого-нибудь действия, кроме отображения информации. Метод отправки помещает значения, введенные в форме, в переменную `$_REQUEST` языка PHP. Зачастую при обработке формы в Joomla методом отправки используется компонент, выдающий конечный результат такой обработки.

За исключением чтения, функции CRUD (создания, чтения, обновления и удаления) внедрены непосредственно в код компонентов для добавления, обновления и удаления строк из таблицы. И, как правило, эти функции CRUD в Joomla выполняют компоненты.

Своеобразие компонентов

Подводя краткий итог всему сказанному выше, следует заметить, что компоненты являются основными стандартными блоками для построения веб-сайтов на Joomla. Ниже перечислены их основные свойства.

- Являются основными стандартными блоками для построения веб-сайта на Joomla.
- Служат для сопровождения как содержимого, так и пунктов меню веб-сайта.
- Служат, как правило, точкой входа в цикл обработки данных в Joomla.
- Применяются в административной части веб-сайта для управления всем процессом его администрирования.
- Служат для выполнения всех функций CRUD при обращении к базе данных.
- Служат, как правило, для обработки любых форм, где применяется метод отправки.

Шаблон проектирования MVC

Прежде чем переходить к подробному рассмотрению основного компонента, сделаем краткий обзор шаблона проектирования MVC, применяемого во всех компонентах. Как и во всех остальных шаблонах проектирования, предназначенных для объектно-ориентированного программирования (ООП), в шаблоне MVC предпринимается попытка извлечь наибольшую выгоду из повторно используемого кода (классов и методов), разделяя обработку на требующиеся разнотипные задания и создавая отдельные классы для их выполнения. Кроме того, применение шаблона MVC упрощает чтение, понимание и сопровождение самого кода. Те, у кого имеется некоторый опыт применения шаблонов, могут быстро уяснить организацию кода.

Рассмотрим типичный сеанс работы в Joomla, где вводится статья. Ниже приводится краткая сводка наступающих при этом событий.

Пользователь:	Щелкает на кнопке Add (Добавить) на экране Article Manager (Диспетчер статей).
Joomla:	1. Обрабатывает событие, наступающее после щелчка на кнопке, и передает URL для обращения к пустой форме.
	2. Обрабатывает URL для обращения к пустой форме и отображает ее.
Пользователь:	Вводит информацию и щелкает на кнопке Save & Close (Сохранить и закрыть).
Joomla:	1. Обрабатывает форму.
	2. Сохраняет статью и связанную с ней информацию в базе данных.
	3. Создает URL для обращения к компоненту Article Manager с сообщением об успешном завершении операции.
	4. Обрабатывает URL для отображения на экране Article Manager сообщения об успешном завершении операции.

Шаблон проектирования MVC разделяет все эти задачи программирования на три группы. В частности, контроллер обрабатывает вводимые пользователем данные. Модель организует взаимодействие с базой данных, имея некоторые "знания" о том, как действуют статьи. А представление отображает все содержимое на странице. Организуя код подобным образом, мы можем вносить изменения в одну часть компонента, не меняя все остальные. В частности, мы могли бы добавить новое представление, не изменяя контроллер или модель. Если нам потребуется ввести новое действие пользователя (например, копирование), мы можем сделать это в контроллере и модели, не видоизменяя уже имеющийся код. А если нам нужно изменить порядок действия статей (например, добиться наличия в них лишь минимальных значений метаданных), то для этого придется внести изменения только в модель, никак не затрагивая контроллер и представление.

Принимая во внимание все сказанное выше, можно было бы предположить, что в Joomla компонент должен состоять всего из трех файлов: модели, контроллера и представления. Но, к сожалению, на практике дело обстоит много сложнее, и, как будет показано в следующем разделе, чем сложнее компонент, тем больше у него файлов. Тем не менее все эти файлы разделяются в основном на три категории: модели, контроллера и представления, а следовательно, шаблон MVC помогает лучше организовать код и уяснить его.

Административная часть компонента Weblinks

Начнем рассмотрение компонентов с того, как базовый компонент Weblinks действует в административной части веб-сайта. Этот компонент, типичный для данной части веб-сайта, очень похож на другие ее компоненты, предназначенные для управления статьями, пользователями, модулями и т.д.

В табл. 7.1 перечислены программные файлы данного компонента. Все эти файлы находятся в папке `administator/components/com_weblinks`. В остальной части этой главы имена всех файлов компонента Weblinks упоминаются относительно данной начальной папки, если не указано иное. Большинство файлов организовано по шаблону MVC. В частности, все файлы представлений находятся в подпапке представлений, а все основные файлы моделей и контроллеров — в подпапках моделей и контроллеров соответственно. Их также дополняют установочные, конфигурационные и вспомогательные файлы.

Таблица 7.1. Файлы административной части компонента Weblinks, кроме файлов `index.html`

Имя файла	Описание	Категория по шаблону MVC
`controllers/weblink.php`	Основной контроллер для редактирования одиночной веб-ссылки	Контроллер
`controllers/weblinks.php`	Основной контроллер для составления и отображения списка веб-ссылок на экране Weblinks Manager (Диспетчер веб-ссылок)	Контроллер
`helpers/weblinks.php`	Предоставляет различные методы, применяемые в контроллерах и представлениях	Разное
`models/fields/ordering.php`	Предоставляет специальный класс `JformField` для отображения столбца упорядочения веб-ссылок на экране Weblinks Manager	Модель
`models/forms/weblink.xml`	XML-файл, используемый в классе `JformField`, для разметки формы с полями ввода и редактирования веб-ссылок на экране	Модель
`models/weblink.php`	Модель для экранной формы одиночной веб-ссылки	Модель
`models/weblinks.php`	Модель для экранной формы диспетчера веб-ссылок	Модель
`sql/install.mysql.utf8.sql`	SQL-файл для составления таблицы веб-ссылок во время установки	Установка
`sql/uninstall.mysql.ut8.sql`	SQL-файл для удаления таблицы веб-ссылок во время установки	Установка
`tables/weblink.php`	Предоставляет класс `WeblinksTableWeblink`	Модель
`views/weblink/tmpl/edit_metadata.php`	Файл исходной компоновки для редактирования метаданных веб-ссылки	Представление

Окончание табл. 7.1

Имя файла	Описание	Категория по шаблону MVC
views/weblink/tmpl/edit_params.php	Файл исходной компоновки для редактирования вариантов выбора одиночной веб-ссылки	Представление
views/weblink/tmpl/edit.php	Файл исходной компоновки для редактирования веб-ссылки	Представление
views/view.html.php	Класс основного представления для вывода одиночной веб-ссылки в формате HTML	Представление
views/weblinks/view.html.php	Файл исходной компоновки для диспетчера веб-ссылок	Представление
views/weblinks/view.html.php	Класс основного представления для вывода веб-ссылок в формате HTML непосредственно на экране Weblinks Manager	Представление
access.xml	XML-файл, предоставляющий список действий для системы списков контроля доступа (ACL)	Файл com_config
config.xml	XML-файл, предоставляющий список вариантов выбора конфигурации компонента	Файл com_config
controller.php	Класс основного контроллера	Контроллер
weblinks.php	Точка входа для запроса	Контроллер
weblinks.xml	XML-файл, используемый для управления процессом установки	Установка

Установочные файлы

При установке или удалении компонента Weblinks выполняются три файла. Два из них содержат сценарии базы данных SQL, которые выполняются программой установки и удаления компонента. В данном случае по сценарию из файла sql/install.mysql.utf8.sql создается таблица #__weblinks, а по сценарию из файла sql/uninstall.mysql.ut8.sql эта таблица удаляется (или *опускается* в терминологии баз данных). Для некоторых компонентов выполнение сценария базы данных SQL может потребоваться, чтобы заполнить одни таблицы данными из других таблиц. В таком случае подобный сценарий включается в установочный сценарий. А файл weblinks.xml выполняет ту же функцию, что и XML-файлы, созданные ранее в примерах модулей и подключаемых модулей. Он содержит те же самые элементы разметки для автора, даты создания и прочего, а также список файлов и поддерживаемых языков.

Следует иметь в виду, что все основные компоненты предварительно устанавливаются при первой установке Joomla. В установочной папке находится файл installation/sql/mysql/joomla.sql, содержащий сценарий для заполнения таблиц базы данных всеми основными расширениями. Сценарий базы данных SQL из второго файла, installation/sql/mysql/sample_data.sql, выполняется в том случае, если устанавливаются используемые по умолчанию выборочные данные. Тем не менее многие основные расширения могут быть удалены с помощью компонента Extension Manager (Диспетчер расширений). В этом случае установочный XML-файл служит для управления процессом удаления.

Меню компонентов

Для управления компонентом при его создании обычно требуется один или более пункт меню в административной части веб-сайта. Меню административной части веб-сайта на Joomla создаются иначе, чем меню пользовательской части. Пункты меню в пользовательской части определяются администратором веб-сайта с помощью административного компонента com_menus. А в административной части пункты добавляются в меню Components при установке компонентов. Эти пункты меню определяются в XML-файле компонента. Для этой цели в элементе разметки administration из файла weblinks.xml имеется, например, следующий код:

```
<menu img="class:weblinks">Weblinks</menu>
  <submenu>
  <!--
    Все знаки & должны быть экранированы знаками &amp; чтобы XML-файл
    был признан действительным и синтаксически проанализирован установщиком
-->
    <menu link="index.php?option=com_weblinks" view="links"
↳img="class:weblinks"
      alt="Weblinks/Links">Links</menu>
    <menu link="index.php?option=com_categories&amp;extension=com_weblinks"
      view="categories" img="class:weblinks-cat"
↳alt="Weblinks/Categories">Categories
    </menu>
</submenu>
```

Приведенные выше элементы разметки menu используются для ввода строк в таблицу #__menus при установке компонента Weblinks. А из этих строк формируются пункты меню Components в административной части веб-сайта на Joomla, как показано на рис. 7.1.

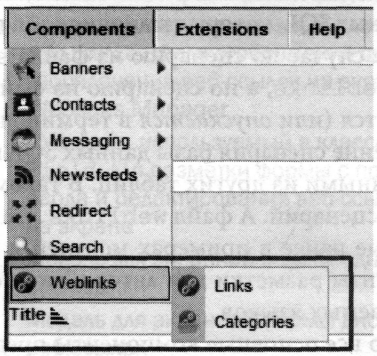


Рис. 7.1. Пункты меню компонента Weblinks

Более подробно формат элементов разметки menu будет рассматриваться в главе 9, когда мы будем создавать свой собственный компонент.

Параметры компонента

Параметры компонента сложнее, чем у других расширений. При создании или редактировании подключаемых модулей, шаблонов, модулей и языковых расширений содержимое элемента разметки `config` считывается из XML-файла с целью определить параметры данного расширения. Набор параметров определяется для каждого экземпляра расширения.

Параметры компонентов могут быть определены на двух или более уровнях. Глобальные параметры могут быть заданы при выборе пиктограммы Options на панели инструментов в окне диспетчера соответствующего компонента. Глобальные параметры используются в базовых компонентах для установки по умолчанию вариантов выбора новых пунктов меню. В ходе этого процесса вызывается административный компонент `com_config`, использующий два XML-файла, `config.xml` и `access.xml` для создания режимного окна с параметрами компонента. Каждому элементу разметки `fieldset` из файла `config.xml` соответствует отдельная вкладка в окне с параметрами компонента. Последний элемент разметки `fieldset` принято называть `permissions` (полномочия), причем он содержит один элемент разметки `field`, как показано ниже.

```
<field name="rules" type="rules"
    component="com_weblinks"
    filter="rules"
    validate="rules"
    label="JCONFIG_PERMISSIONS_LABEL"
    section="component" />
```

Более подробно этот вопрос будет рассматриваться в главе 10 при создании пользовательского компонента.

Вспомогательные классы и методы

В административной части компонента Weblinks применяются два вспомогательных метода из файла `helpers/weblinks.php`. Вспомогательные классы служат для хранения небольших методов, которые могут быть использованы в разных местах компонента и не вписываются в модель, представление или сам компонент. В частности, вспомогательный метод `addSubmenu()` добавляет подменю, отображаемое на экране Weblinks Manager, как показано на рис. 7.2.

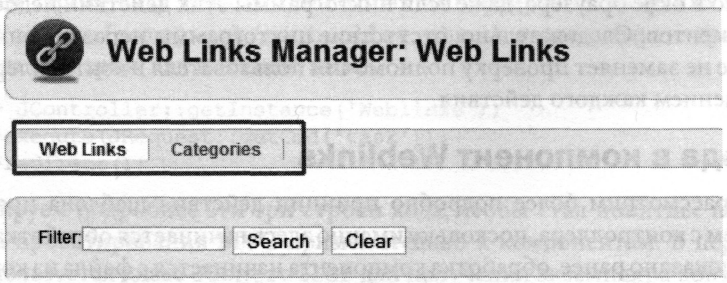


Рис. 7.2. Подменю компонента Weblinks

Эти подменю не следует путать с подменю, доступными из меню Components, приведенного на рис. 7.1. Те подменю создаются на основании XML-файла компонента во время его установки, а эти воспроизводятся как ссылки на экране Weblinks Manager, обеспечивая переход между экранами двух диспетчеров: веб-ссылок и категорий.

Второй вспомогательный метод, `getActions()`, служит для определения тех действий, которые пользователю разрешено производить над компонентом. Для этой цели на экране Weblinks Manager имеется панель инструментов, приведенная на рис. 7.3. А на экране Weblinks Manager: Weblink имеется своя панель инструментов, приведенная на рис. 7.4.



Рис. 7.3. Вид панели инструментов на экране Weblinks Manager



Рис. 7.4. Вид панели инструментов на экране Weblinks Manager: Weblink

Если у пользователя нет полномочий на выполнение одного или более действия, пиктограмма соответствующего действия отсутствует на панели инструментов. Вспомогательный метод `getActions()` возвращает объект со списком разрешенных действий над компонентом, а также разрешениями на их выполнение. Эти сведения используются в представлениях для определения тех панелей инструментов, которые должны быть отображены.



Безусловно, отображение на панелях инструментов только тех действий, которые разрешено выполнять пользователям, является удобным свойством пользовательского интерфейса. Но не следует забывать, что оно никак *не* защищает от попыток злоумышленника выполнить неразрешенное действие. Ведь пользователь может попытаться выполнить нужные ему действия, отредактировав переменную `$_REQUEST` языка PHP непосредственно в окне браузера, даже если пиктограммы этих действий недоступны на панелях инструментов. Следовательно, отсутствие пиктограммы неразрешенного действия ни в коей мере не заменяет проверку полномочий пользователя в контроллере или модели перед выполнением каждого действия.

Точка входа в компонент Weblinks

А теперь рассмотрим более подробно принцип действия шаблона проектирования MVC. И начнем с контроллера, поскольку именно здесь начинается обработка компонента.

Как было показано ранее, обработка компонента начинается с файла из корневой папки этого компонента с тем же именем, что и у компонента, но без префикса `com_`. Этот файл называется точкой входа в компонент. Так, экран Weblinks Manager: Weblink открывается по следующему URL: `administrator/index.php?option=com_weblinks&view=weblinks`.

А обработка этого URL начинается в файле `weblinks.php`, находящемся в папке `administrator/com_weblinks/`. Содержимое этого файла приведено в листинге 7.1.

Листинг 7.1. Исходный код из файла `administrator/com_weblinks/weblinks.php`

```
<?php
/**
 * @package      Joomla.Administrator
 * @subpackage   com_weblinks
 * @copyright    Copyright (C) 2005 - 2012 Open Source Matters, Inc. All
↪rights reserved.
 * @license      GNU General Public License version 2 or later; see LICENSE.
txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

// Проверка прав доступа.
if (!JFactory::getUser()->authorise('core.manage', 'com_weblinks')) {
    return JError::raiseWarning(404, JText::_('JERROR_ALERTNOAUTHOR'));
}

// включить зависимости
jimport('joomla.application.component.controller');

$controller = JController::getInstance('Weblinks');
$controller->execute(JRequest::getCmd('task'));
$controller->redirect();
```

В этом файле содержится сценарий PHP, но не объявляются классы, и поэтому он выполняется сразу же после загрузки. После оператора `defined` в сценарии из этого файла сразу же выполняется проверка полномочий пользователя на выполнение действия `core.manage` над компонентом Weblinks. Действие `core.manage` требуется выполнить перед тем, как пользователь сможет приступить к решению любых задач административного управления основными компонентами. Благодаря этой первоначальной проверке предотвращается несанкционированный доступ пользователя к экранной форме компонента Weblinks путем непосредственного ввода URL в окне браузера.

В следующей строке кода класс `JController` импортируется из файла `libraries/joomla/application/component/controller.php`. А далее следуют три завершающие строки кода сценария, как показано ниже.

```
$controller = JController::getInstance('Weblinks');
$controller->execute(JRequest::getCmd('task'));
$controller->redirect();
```

Проанализируем подробнее эти три строки кода, чтобы стал понятнее принцип действия шаблона проектирования MVC применительно к компонентам. В первой из этих строк кода используется класс `JController` для получения экземпляра контроллера, необходимого для выполнения текущей задачи. Во второй строке кода объект этого контроллера используется для выполнения задачи. А в третьей строке кода вызывается метод `redirect()` данного контроллера для перехода по следующему URL, если таковой имеется.

Те же самые строки кода содержатся в первоначальном сценарии *практически всех* базовых компонентов. И если понять принцип их действия в компоненте `com_weblinks` в частности, это послужит отличной отправной точкой для уяснения роли шаблона MVC в Joomla в общем.

Функции контроллера в компоненте Weblinks

Рассмотрим действие этого механизма, когда пользователь выполняет типичные действия на экране Weblinks Manager в административной части веб-сайта на Joomla.

Пример 1. Пользователь выбирает пункт меню Components⇨Weblinks

В данном примере пользователь выбирает пункт Weblinks из меню Components на экране Weblinks Manager. Для перехода по этому пункту меню служит следующий URL: `administrator/index.php?option=com_weblinks`. В первой из рассматриваемых здесь строк кода метод `getInstance()` обнаруживает класс контроллера, подходящий для выполнения указанной задачи. Но в данном примере конкретная задача не указана, и поэтому решается задача по умолчанию, а именно: отображение информации. Для выполнения задачи отображения информации используется базовый контроллер `WeblinksController` компонента Weblinks из файла `controller.php`.

Во второй строке кода для выполнения указанной задачи выполняется метод `execute()`. Но поскольку конкретная задача не указана, то по умолчанию выполняется задача отображения информации. С этой целью вызывается метод `display()` из класса `WeblinksController`. Он отображает экран Weblinks Manager, но не устанавливает значение переадресации.

В третьей строке кода выполняется метод `redirect()`. Он просто возвращает логическое значение `false` и отображает экран диспетчера данного компонента, поскольку переадресация не указана.

Ниже приведены главные составляющие действия в данном примере.

- URL: `administrator/index.php?option=com_weblinks`.
- Задача: не указана (по умолчанию решается задача отображения информации).
- Контроллер: класса `WeblinksController` из файла `controller.php`.
- Выполняемый метод контроллера: `display()`.
- Переадресация контроллера: отсутствует.

Пример 2. Пользователь щелкает на названии веб-ссылки, чтобы отредактировать ее

Ниже приведены исходные данные для первой части данного примера.

- URL: `administrator/index.php?option=com_weblinks&task=weblink.edit&id=7`.
- Задача: `weblink.edit`.
- Контроллер: класса `WeblinksControllerWeblink` из файла `controllers/weblink.php`.
- Выполняемый метод контроллера: `JControllerForm->edit()`.
- Переадресация контроллера: `index.php?option=com_weblinks&view=weblink&layout=edit&id=7`.

В самом URL указывается решаемая задача: `weblink.edit`, т.е. редактирование вебссылки. Этим методу `getInstance()` из класса `JController` предписывается найти контроллер с именем `WeblinksControllerWeblink`. Его имя составляется по следующему образцу:

```
<имя компонента> + Controller + <первая часть названия задачи>
```

Метод `getInstance()` из класса `JController`, находящегося в файле `libraries/joomla/application/component/controller.php`, содержит следующий код для обработки указанной задачи:

```
// проверить команду контроллер.задача
if (strpos($command, '.') !== false) {
    // расчленить команду контроллер.задача
    list($type, $task) = explode('.', $command);

    // определить имя файла контроллера и путь к нему
    $file = self::createFileName('controller', array('name' => $type,
'format' => $format));
    $path = $basePath.'/controllers/'.$file;

    //  восстановить задачу без контекста контроллера
    JRequest::setVar('task', $task);
```

В приведенном выше фрагменте кода название задачи расчленяется на части, разделяемые точкой. В данном примере задача `weblink.edit` превращается в массив, где `weblink` — тип задачи, а `edit` — ее название. Тип задачи становится третьей частью имени класса контроллера (в данном случае — `Weblink` в `WeblinksControllerWeblink`). Затем переменная запроса на обработку задачи изменяется таким образом, чтобы в нее была включена лишь вторая часть названия задачи. В данном примере в переменную `$_REQUEST['task']` включается лишь часть `edit` из названия задачи `weblink.edit`.

Когда черед доходит до строки кода `$controller->execute()`, выполнение задачи редактирования веб-ссылки поручается классу `WeblinksControllerWeblink`. У этого класса отсутствует собственный метод `edit()`, и поэтому он вызывает метод из своего родительского класса `JControllerForm`, находящегося в файле `libraries/joomla/application/component/controllerform.php`. В этом методе проверяется, разрешено ли текущему пользователю редактировать данный элемент и снят ли он с регистрации для другого пользователя. Если все нормально, то идентификатор данного элемента сохраняется в переменной сеанса работы пользователя. Он используется на следующей стадии обработки, когда происходит переадресация и фактическая загрузка элемента для редактирования.

И на последней стадии устанавливается и происходит переадресация по новому URL, используемому в Joomla для загрузки редактируемого элемента. В данном случае переадресация происходит по следующему URL:

```
index.php?option=com_weblinks&view=weblink&layout=edit&id=7
```

Слово `administrator` добавляется в начале URL при обработке методом `redirect()`, поскольку действие происходит в административной части веб-сайта.

Обработкой URL для переадресации начинается второй цикл запроса. Ниже приведены исходные данные для этой обработки.

- URL: `administrator/index.php?option=com_weblinks&view=weblink&layout=edit&id=7`
- Задача: не указана (по умолчанию решается задача отображения информации).
- Контроллер: класса `WeblinksController` из файла `controller.php`.
- Выполняемый метод контроллера: `WeblinksController->edit()`.
- Переадресация контроллера: отсутствует.

Указанный выше URL не содержит задачу, и поэтому снова используется базовый класс контроллера `WeblinksController` и выполняется метод `display()`. Этот метод содержит следующий код:

```
$view = JRequest::getCmd('view', 'weblinks');
$layout = JRequest::getCmd('layout', 'default');
$id = JRequest::getInt('id');

// проверить наличие формы редактирования
if ($view == 'weblink' && $layout == 'edit' &&
↳!$this->checkEditId('com_weblinks.edit.weblink', $id)) {

    // не разрешать несанкционированный доступ к форме
    $this->setError(JText::sprintf('JLIB_APPLICATION_ERROR_UNHELD_ID',
↳$id));
    $this->setMessage($this->getError(), 'error');
    $this->setRedirect(
↳JRoute::_('index.php?option=com_weblinks&view=weblinks', false));

    return false;
}
```

Напомним, что в предыдущем цикле запроса идентификатор редактируемого элемента был сохранен в переменной сеанса работы пользователя. Поэтому в методе `checkEditId()` текущий идентификатор проверяется на совпадение с сохраненным ранее. А для чего это нужно делать?

В предыдущем цикле запроса было проверено, что пользователю разрешено редактировать данный элемент и что он еще не снят с регистрации для другого пользователя. Повторять все эти проверки нецелесообразно, но в то же время нужно уберечься от попыток злоумышленника отредактировать данный элемент, введя непосредственно такой же URL, как и тот, что использовался для загрузки формы редактирования. Дополнительная проверка данных о сеансе работы пользователя, сохраненных в предыдущем цикле запроса, позволяет еще раз убедиться в том, что это тот же самый, уже проверенный ранее пользователь.

Если все нормально, то выполнение метода `display()` из класса `WeblinksController` продолжается далее. На этот раз указаны представление и компоновка, и поэтому форма редактирования отображается вместо экранной формы диспетчера. Эта форма отображается еще и потому, что переадресация отсутствует. Таким образом, выполняется один цикл запроса для проверки полномочий пользователя на редактирование данного

элемента. При этом формируется URL для переадресации к форме редактирования с целью ее последующего отображения.

Пример 3. Пользователь щелкает на кнопке Save & Close в форме редактирования

Ниже приведены главные составляющие действия в данном примере.

- URL: `administrator/index.php?option=com_weblinks&layout=edit&id=7`.
- Задача: `weblink.save`.
- Контроллер: класса `WeblinksControllerWeblink` из файла `controllers/weblink.php`.
- Выполняемый метод контроллера: `JControllerForm->save()`.
- Переадресация контроллера: `administrator/index.php?option=com_weblinks&view=weblinks`.

В данном примере URL не содержит часть, в которой указана задача. Вместо этого задача указывается в форме редактирования. В частности, разметка кнопки Save & Close содержит следующий атрибут:

```
onclick="javascript:Joomla.submitbutton('weblink.save')"
```

Если щелкнуть на этой кнопке, начнет выполняться функция языка JavaScript, устанавливающая значение `"weblink.save"` в поле `task` формы редактирования и передающая форму на обработку. В итоге задача помещается в массив запроса.

Как и прежде, третья часть имени класса контроллера извлекается из первой части названия задачи, в результате чего получается следующее имя: `WeblinksControllerWeblink`. В этом классе контроллера далее выполняется метод `save()`. Но поскольку у этого класса нет собственного метода `save()`, то он вызывает его из своего родительского класса `JControllerForm`.

Для данного контроллера получается класс модели и снова проверяется совпадение идентификатора редактируемого элемента с сохраненным в сеансе работы пользователя. После этого проверяется, разрешено ли пользователю сохранять данный элемент. Вслед за этим из загруженного класса модели вызывается метод `validate()` для проверки достоверности введенных данных, которые затем сохраняются с помощью метода `save()`. Если все эти действия выполнены успешно, формируется сообщение об успешном сохранении отредактированного элемента.

И на последней стадии устанавливается переадресация, которая зависит от конкретного действия: сохранение (**Save**), сохранение и закрытие (**Save & Close**) или сохранение и создание (**Save & New**). Если выполняется сохранение и закрытие, то с редактируемого элемента снимается блокировка и переадресация устанавливается на загрузку экрана Weblinks Manager. В этом случае переадресация происходит в том же самом цикле, что и в рассмотренном выше примере 1.

Разделяя задачу сохранения на два цикла запроса: один — для действия сохранения, а другой — для переадресации, можно воспользоваться одним и тем же методом `save()`, чтобы выполнить разные действия сохранения. Первый цикл запроса одинаков для всех этих действий. Отличия лишь в том, куда именно переадресуется пользователь. Так,

действие сохранения оставляет пользователя в текущей экранной форме редактирования, а действие сохранения и закрытия осуществляет переход пользователя к новой пустой экранной форме. В данном примере действие сохранения и закрытия возвращает пользователя обратно к экрану Weblinks Manager.

Пример 4. Пользователь удаляет некоторые веб-ссылки

В данном примере пользователь устанавливает флажок рядом с названиями одного или более элемента веб-ссылок и щелкает на пиктограмме корзины (кнопке Trash), находящейся на панели инструментов экрана Weblinks Manager. Ниже приведены главные составляющие действия в данном примере.

- URL: `administrator/index.php?option=com_weblinks&view=weblinks`.
- Задача: `weblinks.trash`.
- Контроллер: класса `WeblinksControllerWeblink` из файла `controllers/weblink.php`.
- Выполняемый метод контроллера: `JControllerAdmin->publish()`.
- Переадресация контроллера: `administrator/index.php?option=com_weblinks&view=weblinks`.

В данном примере URL просто возвращает обратно к экрану Weblinks Manager. Задача `weblinks.trash` исходит из сценария JavaScript, который запускается на выполнение из атрибута `onclick` в элементе привязки `a` для разметки пиктограммы корзины на панели инструментов. Ниже приведен код разметки этого атрибута упомянутым сценарием.

```
onclick="javascript:
if (document.adminForm.boxchecked.value==0){
    alert('Please first make a selection from the list');
} else {
    Joomla.submitbutton('weblinks.trash')
}"
```

Панели инструментов в экранных формах административной части веб-сайта предоставляются административным модулем `mod_toolbar`. Код сценария JavaScript для пиктограммы корзины генерируется автоматически в классе `JButtonStandard` из файла `libraries/joomla/html/toolbar/button/standard.php`. Этот сценарий действует таким же образом, как и сценарий JavaScript из предыдущего примера, за исключением того, что в нем сначала проверяется выбор хотя бы одного элемента для удаления. И в этом случае в элемент `task` разметки формы вводится значение `"weblinks.trash"`, после чего форма передается на обработку.

Первой частью названия задачи является `weblinks`, и поэтому класс соответствующего контроллера называется `WeblinksControllerWeblinks`. И хотя второй частью названия задачи является `trash`, на самом деле выполняется метод `publish()` из класса этого контроллера. Каким же образом из задачи `trash` вытекает вызов метода `publish()`?

Сопоставление задач с методами

Ответ на поставленный выше вопрос состоит в том, что для сопоставления задач с методами в классе `JController` используется поле массива под названием `$taskMap`. Это

поле создается в конструкторе данного класса контроллера. А при выполнении метода execute() из класса JController осуществляется поиск задачи в массиве $taskMap и последующее выполнение соответствующего метода из класса контроллера для этой задачи. Рассмотрим подробнее, как это происходит. Но предварительно следует заметить, что если вы незнакомы с конструкторами и их особой ролью в ООП, обратитесь за справкой к приведенной ниже врезке "Методы-конструкторы".

Методы-конструкторы

Начинающим осваивать объектно-ориентированное программирование (ООП) очень важно уяснить особую роль метода-конструктора. Конструктор — это метод, который выполняется автоматически при создании объекта класса с помощью оператора new. Например, в строке кода

```
$registry = new JRegistry();
```

создается новый объект класса JRegistry и выполняется метод-конструктор этого класса перед присваиванием его объекта переменной $registry.

В языке PHP конструктор называется __construct. Он служит для инициализации значений объекта и принимает аргументы, но не возвращает никаких значений. Из одного конструктора можно вызвать другой конструктор родительского класса, используя следующий синтаксис:

```
parent::__construct();
```

Если в классе отсутствует метод-конструктор, то вызывается конструктор родительского класса, как это обычно происходит с методами, но при условии, что конструктор определен в родительском классе. В более поздних версиях языка PHP конструкторы стали называться по имени своих классов, например, конструктор jregistry() класса JRegistry вместо __construct(). Если же метод __construct() отсутствует в коде PHP, то вместо него будет произведен поиск метода-конструктора по имени класса. А в Joomla все конструкторы называются __construct().

Ниже приведены первые строки кода из конструктора класса JControllerAdmin.

```
public function __construct($config = array())
{
    parent::__construct($config);
```

В этом фрагменте кода вызывается конструктор родительского класса JController. В этом конструкторе применяется способ так называемой рефлексии для построения массива с именами всех открытых в данном классе методов, как следует из приведенного ниже кода.

```
// определить методы, исключаемые из базового класса
$xMethods = get_class_methods('JController');

// получить открытые в данном классе методы, используя рефлексию
$r = new ReflectionClass($this);
$rName = $r->getName();
$rMethods = $r->getMethods(ReflectionMethod::IS_PUBLIC);
$methods = array();

foreach ($rMethods as $rMethod)
```

```
{
    $mName = $rMethod->getName();

    // добавить вызываемый по умолчанию метод отображения, если
    // он не объявлен явным образом
    if (!in_array($mName, $xMethods) || $mName == 'display') {
        $this->methods[] = strtolower($mName);
        // автоматически зарегистрировать методы как задачи
        $this->taskMap[strtolower($mName)] = $mName;
    }
}
```

В приведенном выше фрагменте кода поле `taskMap` заполняется ассоциативным массивом, где ключи и значения содержат имена открытых методов в данном классе, за исключением методов из его родительского класса `JController`. Кроме того, всегда добавляется вызываемый по умолчанию метод отображения, поскольку он присутствует в классе `JController`. Если вы незнакомы с понятиями открытых, закрытых и защищенных методов и полей, обратитесь за справкой к приведенной далее врезке "Модификаторы доступа `public`, `protected` и `private`".

Через несколько строк кода в этом же методе следует приведенный ниже фрагмент кода.

```
// если исходная задача определена, зарегистрировать ее как таковую
if (array_key_exists('default_task', $config)) {
    $this->registerDefaultTask($config['default_task']);
}
else {
    $this->registerDefaultTask('display');
}
```

В этом фрагменте кода вводится исходная (по умолчанию) задача с ключом `__default` и значением `"display"`. Этим Joomla уведомляется, что нужно делать с запросом, если конкретная задача не указана. Для основных компонентов это, как правило, задача отображения.

Рассмотрим, как этот механизм действует на данном конкретном примере. При вызове метода `JController::getInstance()` создается новый класс контроллера `WeblinksControllerWeblinks`. А поскольку у него отсутствует конструктор, то вызывается конструктор его родительского класса `JControllerAdmin`. Как было показано выше, в первой строке кода вызывается конструктор родительского класса `JControllerAdmin`, создающий массив `taskMap`. В классе `WeblinksControllerWeblinks` определен только один метод — `getModel()`. Но поскольку этот же метод присутствует в классе `JController`, он исключается из массива `taskMap`.

В родительском классе `JControllerAdmin` определены следующие методы: `checkin()`, `delete()`, `display()`, `publish()`, `reorder()`, а также `saveorder()`. Из них только метод `display()` определен также в классе `JController`, и поэтому он явным образом вводится в массив, а все остальные методы — обычным образом. Кроме того, конструктор класса `JController` добавляет в массив элемент с ключом `__default`, имеющим по умолчанию значение `"display"`. По завершении выполнения конструктора из родительского класса `JController` в массиве `taskMap` содержатся значения, приведенные в табл. 7.2.

Таблица 7.2. Значения, хранящиеся в массиве `taskMap` после выполнения конструктора из класса `JController`

Ключ (задача)	Значение (исполняемый метод)
delete	delete
display	display
publish	publish
reorder	reorder
saveorder	saveorder
checkin	checkin
__default	display

Ниже приведена следующая часть кода из конструктора родительского класса `JControllerAdmin`.

```
// определить сопоставления со стандартными задачами
$this->registerTask('unpublish', 'publish'); // значение = 0
$this->registerTask('archive', 'publish'); // значение = 2
$this->registerTask('trash', 'publish'); // значение = - 2
$this->registerTask('report', 'publish'); // значение = - 3
$this->registerTask('orderup', 'reorder');
$this->registerTask('orderdown', 'reorder');
```

Именно здесь дополнительные задачи сопоставляются с соответствующими методами. Например, в выделенной выше строке кода задача `trash` сопоставляется с методом `publish()` добавлением в массив `taskMap` элемента с ключом `trash` и значением `"publish"`. После выполнения этого фрагмента кода в массиве `taskMap` появятся значения, приведенные в табл. 7.3.

Таблица 7.3. Значения, хранящиеся в массиве `taskMap` после регистрации задач с помощью метода `registerTask()`

Ключ (задача)	Значение (исполняемый метод)
delete	delete
display	display
publish	publish
reorder	reorder
saveorder	saveorder
checkin	checkin
__default	display
unpublish	publish
archive	publish
trash	publish
report	publish
orderup	reorder
orderdown	reorder

Этот способ сочетает в себе простоту и гибкость. Если правильно назвать задачи по именам соответствующих методов контроллера, то для их сопоставления не придется больше ничего делать, поскольку все остальное возьмет на себя Joomla. А если потребуется видоизменить сопоставление, для этого достаточно воспользоваться методом `registerTask()`, как показано выше.

В данном примере имеет смысл воспользоваться тем же самым методом для выполнения задач публикации (`publish`), снятия публикации (`unpublish`), архивирования (`archive`), удаления в корзину (`trash`) и составления отчета (`report`). Все эти задачи просто изменяют значение в столбце `state` (состояние) таблицы `#__weblinks`.

Модификаторы доступа `public`, `protected` и `private`

Начинающим осваивать ООП очень важно уяснить, чем отличаются друг от друга открытые, закрытые и защищенные методы и поля. Открытые методы, объявленные с модификатором доступа `public`, можно вызывать из любого класса или сценария; защищенные методы, объявленные с модификатором доступа `protected`, — только из экземпляра текущего класса или его подкласса; а закрытые методы, объявленные с модификатором доступа `private`, — только из экземпляра текущего класса.

Для чего требует такое различие методов? Дело в том, что вместе с классом, как правило, создаются некоторые методы, которые предполагается использовать в других классах. И зачастую это главная причина для создания класса. Поэтому такие методы должны быть открытыми, т.е. общедоступными.

Для сохранения компактности и конкретности назначения можно также создать вспомогательные методы, предназначенные для только поддержки основных открытых методов. Но эти методы не должны быть доступны за пределами их класса. Если, например, потребуется реорганизовать вспомогательные методы, не затрагивая код в остальной части программы, их можно объявить закрытыми, чтобы их нельзя было вызывать за пределами текущего класса.

Промежуточное положение занимают защищенные методы. Они могут использоваться в подклассах текущего класса, но не в тех классах, которые с ним не связаны.

Все сказанное выше относится и к полям в классах. Открытые поля доступны для любого класса, закрытые поля — только для экземпляров данного класса, а защищенные поля — только для данного класса и его подклассов.

Если поле или метод создается без модификатора доступа, такой элемент кода считается открытым. Так, если речь идет о прикладном программном интерфейсе (API) для класса, то имеются в виду открытые поля и методы. Ведь вносить изменения в интерфейс API без особой необходимости не стоит. А защищенные и закрытые поля и методы не относятся к API, и поэтому их можно свободно изменять, не затрагивая программы, опирающиеся на этот интерфейс.

Данное ограничение не накладывалось в первых версиях языка PHP, а вместо него были приняты соответствующие условные обозначения имен. Так, если имя поля или метода начиналось со знака подчеркивания (_), такой элемент кода считался закрытым и недоступным для использования за пределами текущего класса. Именно поэтому в исходном коде иногда можно встретить закрытые методы и поля, названные подобным образом.

Применение модели

А теперь рассмотрим метод `publish()` из класса `JControllerAdmin`. Ниже приведена первая часть исходного кода этого метода.

```
function publish()
{
    // проверить наличие поддельных запросов
    JRequest::checkToken() or die(JText::_('JINVALID_TOKEN'));
```



В этом фрагменте кода проверяется, произошел ли переход к данной стадии обработки из формы Joomla. Благодаря этому предотвращаются попытки злоумышленника получить доступ к рассматриваемому здесь методу непосредственно из браузера без предварительной регистрации на веб-сайте Joomla. Далее в этом методе следуют приведенные ниже строки кода.

```
// получить из запроса элементы для публикации
$cid = JRequest::getVar('cid', array(), '', 'array');
$data = array('publish' => 1, 'unpublish' => 0, 'archive'=> 2,
↳'trash' => -2, 'report'=>-3);
$task = $this->getTask();
```

В этом фрагменте кода из запроса получается список идентификаторов. В данном примере эти идентификаторы необходимо изменить таким образом, чтобы они обозначали состояние удаления. Затем нужно построить ассоциативный массив, в котором целые значения из столбца `state` таблицы сопоставляются с задачами. Например, для выполнения задачи `publish` значение изменяется на **1**, а для выполнения задачи **`trash`** — на **-2**. Далее задача получается с помощью метода `getTask()`. Значение, присваиваемое задаче, сохраняется при ее выполнении в приведенном ниже коде метода `execute()` из класса `JController`.

```
public function execute($task)
{
  $this->task = $task;
```

Напомним, что задача была изменена в запросе методом `getInstance()`, в результате чего на момент вызова метода `execute()` осталась лишь вторая ее часть. Так, в начале данного примера задача называлась `weblinks.trash`, а теперь от ее названия осталась лишь вторая часть —`trash`. В приведенном выше фрагменте кода видоизмененная задача (`trash`) сохраняется в поле `$task` объекта класса `JController`, и поэтому она становится доступной с помощью метода `getTask()`, когда в ней возникнет потребность.

В следующей строке кода из метода `publish()` значение нового состояния получается из созданного ранее ассоциативного массива:

```
$value = JArrayHelper::getValue($data, $task, 0, 'int');
```

Теперь известна задача, которую следует выполнить, а также новое значение, которое необходимо установить в столбце `state` таблицы. В приведенном ниже следующем фрагменте кода из метода `publish()` выдается ошибка, если отсутствуют элементы для обработки, а иначе известная задача обрабатывается.

```
if (empty($cid)) {
    JError::raiseWarning(500, JText::_($this->
↳text_prefix.'_NO_ITEM_SELECTED'));
}
else {
    // получить модель
    $model = $this->getModel();
```

Для обработки задачи необходимо вызвать метод `getModel()`, как выделено полужирным в последней строке приведенного выше фрагмента кода. Проанализируем эту строку подробнее, поскольку это очень важно для понимания кода. Прежде всего необходимо выяснить, о каком, собственно, классе идет речь. Несмотря на то что данный код выполняется из класса `JControllerAdmin`, текущим на самом деле является класс `WeblinksControllerWeblinks`, поскольку метод `publish()` отсутствует в классе `JControllerAdmin`, а следовательно, он наследуется из родительского класса. Таким образом, оператор `$this` в выражении `$this->getModel()` обозначает ссылку на класс `WeblinksControllerWeblinks`.

Существенно, что в шаблоне проектирования MVC контроллеру *известно*, как получить модель для выполняемой задачи. Именно это и делает метод `getModel()`. В действительности, если проанализировать класс `WeblinksControllerWeblinks`, то в нем можно обнаружить *единственный* метод — `getModel()`! Ниже для справки приведен весь код этого класса.

```
class WeblinksControllerWeblinks extends JControllerAdmin
{
/**
 * Замещает получение модели.
 * @since 1.6
 */
public function getModel($name = 'Weblink', $prefix = 'WeblinksModel',
↳$config = array('ignore_request' => true))
  {
    $model = parent::getModel($name, $prefix, $config);
    return $model;
  }
}
```

Этот метод (и класс) лишь предоставляет информацию, необходимую для поиска модели, подходящей для данного контроллера. В данном случае класс модели называется `WeblinksModelWeblink`, где слово `Weblink` второй раз указывается в единственном числе. Это характерный пример того, как ООП вообще и шаблон MVC в частности позволяет использовать код повторно. Единственным отличием данного конкретного контроллера является имя модели, которое ему требуется. Но только это и обеспечивает код из класса `WeblinksControllerWeblinks`, а все остальные используемые в нем методы наследуются из его родительских классов. То же самое справедливо и для большинства других контроллеров из административной части веб-сайта на Joomla. Например, в версии Joomla 2.5 имеется 17 классов, расширяющих класс `JControllerAdmin`, но только 7 из них содержат единственный метод `getModel()`, а все остальные — совсем немного дополнительного кода, используя главным образом код из своих родительских классов.

Вернемся, однако, к методу `publish()` из класса контроллера `JControllerAdmin`. Вслед за получением модели в этом методе следуют приведенные ниже строки кода.

```
// убедиться в том, что идентификаторы являются целочисленными
JArrayHelper::toInteger($cid);

// опубликовать элементы
if (!$model->publish($cid, $value)) {
    JError::raiseWarning(500, $model->getError());
}
```

В первой строке приведенного выше фрагмента кода вызывается метод из класса `JArrayHelper`, чтобы проверить, все ли значения, хранящиеся в массиве, являются целочисленными. Напомним, что эти значения взяты из запроса, и поэтому, следуя принципам безопасного программирования, необходимо принять меры защиты против злоумышленников. Если заранее известно, что значение, поступающее из запроса, должно быть целочисленным, то его рекомендуется приводить к целочисленному типу явным образом. Этим обеспечивается надежная защита, поскольку целые значения не могут содержать специальные символы, используемые для внесения злонамеренного кода. Именно поэтому в практике разработки принято использовать целые значения в качестве идентификационных номеров или признаков состояния "опубликовано" в столбцах таблиц.

Далее следует кодовый блок условного оператора `if`, в котором из модели вызывается метод `publish()`, возвращающий логическое значение `true` при успешном его выполнении. А если он возвратит логическое значение `false`, значит, в модели произошла какая-то ошибка.

Более подробно модели будут рассматриваться далее в этой главе, а вместе с ними и метод `publish()`. А до тех пор следует иметь в виду, что весь описываемый здесь процесс заключается только во взаимодействии контроллера с моделью для выполнения конкретной задачи. Сначала с помощью метода `getModel()` получается подходящая модель, а затем из нее вызывается нужный метод. Контроллеру известно лишь имя модели и метода, вызываемого для выполнения каждой задачи. И если у модели правильное имя и подходящие методы, ее можно видоизменить, не затрагивая ничего в самом контроллере. Зачастую имена методов в модели воспроизводят имена методов в контроллере. И хотя следовать этому правилу совсем не обязательно, оно тем не менее упрощает сопровождение кода.

Далее в методе `publish()`, вызываемом из контроллера, следуют приведенные ниже строки кода.

```
else {
    if ($value == 1) {
      $ntext = $this->text_prefix.'_N_ITEMS_PUBLISHED';
    }
    else if ($value == 0) {
      $ntext = $this->text_prefix.'_N_ITEMS_UNPUBLISHED';
    }
    else if ($value == 2) {
      $ntext = $this->text_prefix.'_N_ITEMS_ARCHIVED';
    }
    else {
      $ntext = $this->text_prefix.'_N_ITEMS_TRASHED';
    }
      $this->setMessage(JText::plural($ntext, count($cid)));
    }
}
```

Этот кодовый блок выполняется в том случае, если метод `publish()`, вызываемый из модели, завершается успешно. В этом кодовом блоке формируется сообщение о каждой задаче. Обратите внимание на последнюю строку кода, где вызывается метод

JText::plural(), внедренный в версии Joomla 1.6. Этот метод упрощает вывод сообщений, в которых элементы правильно перечисляются во множественном числе на каждом языке.

И в завершающей части метода publish(), вызываемого из контроллера, следуют приведенные ниже строки кода.

```
  $extension = JRequest::getCmd('extension');
  $extensionURL = ($extension) ? '&extension=' .
↳JRequest::getCmd('extension') : '';
  $this->setRedirect(JRoute::_('index.php?option='.
↳$this->option.'&view='.$this->view_list.$extensionURL, false));
}
```

В этом фрагменте кода устанавливается переадресация. В данном примере расширение в запросе отсутствует, и поэтому переадресация берется из пункта меню (значение com_weblinks переменной option) и представления (значение weblinks переменной view_list). В итоге происходит возврат к экрану Weblinks Manager, где появляется сформированное ранее сообщение об успешном выполнении задачи.

Задачи, классы и методы контроллеров компонента Weblinks

Принимая во внимание все сказанное выше, рассмотрим методы из трех контроллеров компонента Weblinks и порядок их сопоставления с задачами этого компонента. Как было показано ранее, в классе WeblinksController из файла controller.php содержится метод display(), выполняющий выбираемую по умолчанию задачу отображения. Это единственная задача, которую выполняет контроллер из данного класса, и только он вообще что-то отображает.

Класс WeblinksControllerWeblinks управляет экраном Weblinks Manager. Он расширяет класс JControllerAdmin, который, в свою очередь, расширяет класс JController. Как пояснялось ранее, в самом классе WeblinksControllerWeblinks содержится единственный метод getModel(). И хотя этот метод открытый, он не включается в массив taskMap, поскольку также находится в классе JController, методы которого исключаются из массива taskMap. В связи с тем что класс JControllerAdmin расширяет класс JController, методы, включаемые в массив taskMap, являются открытыми методами из класса JControllerAdmin, отсутствующими в классе JController.

На рис. 7.5 представлен моментальный снимок экрана из представления Eclipse Type Hierarchy (Иерархия типов в интегрированной среде разработки Eclipse), отображающего наследование класса WeblinksControllerWeblinks от класса JControllerAdmin и методы последнего.

Как видите, открытые методы, обозначенные кружком, используются для выполнения задач. Как пояснялось ранее, в конструкторе класса некоторые дополнительные задачи сопоставляются с методами publish() и saveorder(). Мелкий треугольник слева от метода display() обозначает, что он переопределяет метод отображения из родительского класса. Метод display() вводится в массив taskMap явным образом, хотя он и присутствует в классе JController. Но метод display() из класса JControllerAdmin просто возвращает логическое значение false и больше ничего не делает.

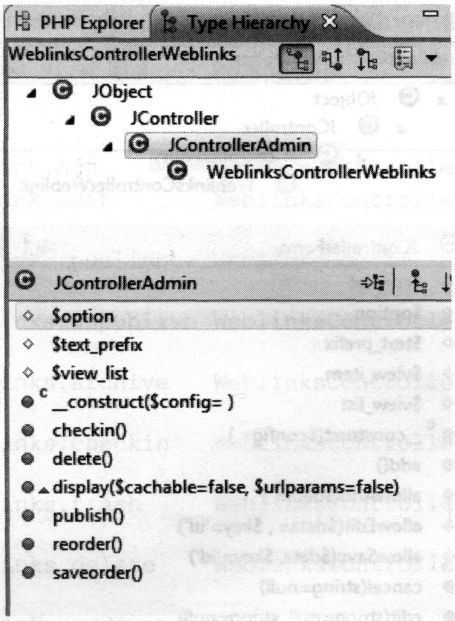


Рис. 7.5. Методы из класса `JControllerAdmin`

Для третьего контроллера имеется класс `WeblinksControllerWeblink` из файла `controllers/weblink.php`. Обратите внимание на то, что в третьей части его имени слово `Weblink` употребляется в *единственном* числе. Этот класс управляет экранной формой ввода и редактирования одиночных веб-ссылок. Такие условные обозначения экранных форм диспетчеров компонентов во множественном числе и форм редактирования одиночных элементов в единственном числе приняты во всех базовых компонентах Joomla, хотя их совсем не обязательно соблюдать, как будет показано в главе 9 при построении собственного компонента.

Сам класс `WeblinksControllerWeblink` содержит лишь два закрытых метода. А методы, сопоставляемые в массиве `taskMap`, происходят из его родительского класса `JControllerForm`. На рис. 7.6 представлен моментальный снимок экрана с иерархией типов и методами данного класса в интегрированной среде разработки Eclipse.

В массив `taskMap` вводятся следующие открытые методы из класса `JControllerForm`: `add()`, `cancel()`, `edit()`, а также `save()`. А в конструкторе этого класса можно обнаружить приведенные ниже строки кода.

```
$this->registerTask('apply', 'save');
$this->registerTask('save2new', 'save');
$this->registerTask('save2copy', 'save');
```

В этих строках кода задачи применения (`apply`), сохранения и создания (`save2new`), а также сохранения в виде копии (`save2copy`) сопоставляются с методом `save()`.

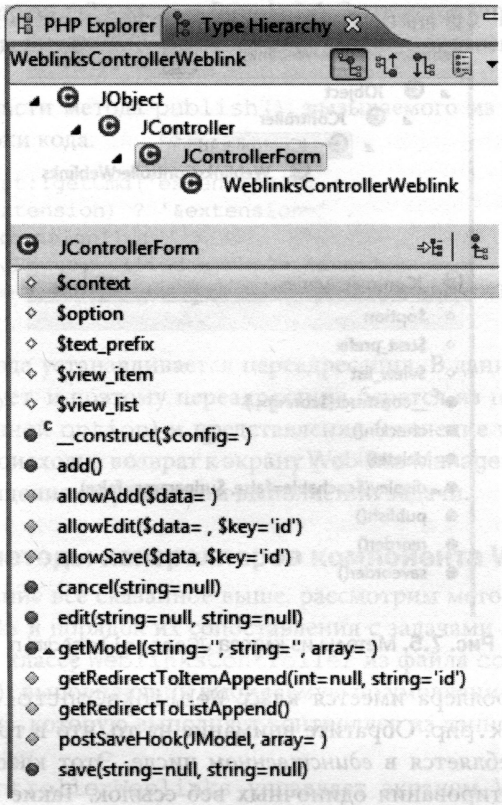


Рис. 7.6. Иерархия типов и методы из класса `JControllerForm`

А теперь подведем краткий итог всем задачам, которые может выполнять пользователь в административной части веб-сайта над компонентом Weblinks. В табл.7.4 приведены все пиктограммы на панели инструментов экрана Weblinks Manager: Weblinks, где перечисляются все веб-ссылки, и соответствующие им задачи, а в табл. 7.5 — пиктограммы на панели инструментов экрана Weblinks Manager: Weblink, где вводятся и редактируются одиночные веб-ссылки, и соответствующие им задачи.

Благодаря этим сведениям можно точно узнать, где именно выполняется каждая задача. Так, если требуется добавить новую задачу, ее можно сопоставить с методом контроллера, уже имеющимся в конструкторе класса, а затем ввести логику в сопоставленный с ней метод или же добавить в класс контроллера новый метод, имя которого совпадает с названием новой задачи. А если потребуется добавить новую экранную форму управления компонентом, то для этой формы можно ввести новый контроллер с его собственными задачами и соответствующими методами.

Таблица 7.4. Задачи, выполняемые на экране Weblinks Manager: Weblinks

Пиктограмма на панели инструментов	Задача	Контроллер	Метод
New (Создать)	weblink.add	WeblinksControllerWeblink	add()
Edit (Редактировать)	weblink.edit	WeblinksControllerWeblink	edit()
Publish (Публиковать)	weblinks.publish	WeblinksControllerWeblinks	publish()
Unpublish (Снять публикацию)	weblinks.unpublish	WeblinksControllerWeblinks	publish()
Archive (Архивировать)	weblinks.archive	WeblinksControllerWeblinks	publish()
Check In (Зарегистрировать)	weblinks.checkin	WeblinksControllerWeblinks	checkin()
Trash (Удалить в корзину)	weblinks.trash	WeblinksControllerWeblinks	publish()
Empty Trash (Очистить корзину)	weblinks.delete	WeblinksControllerWeblinks	delete()
Order Up Arrow (Упорядочить по восходящей)	weblinks.orderup	WeblinksControllerWeblinks	reorder()
Order Down Arrow (Упорядочить по нисходящей)	weblinks.orderdown	WeblinksControllerWeblinks	reorder()
Save Order (Сохранить порядок)	weblinks.saveorder	WeblinksControllerWeblinks	saveorder()

Таблица 7.5. Задачи, выполняемые на экране Weblinks Manager: Weblink

Пиктограмма на панели инструментов	Задача	Контроллер	Метод
Save (Сохранить)	weblink.apply	WeblinksControllerWeblink	save()
Save & Close (Сохранить и закрыть)	weblink.save	WeblinksControllerWeblink	save()
Save & New (Сохранить и создать)	weblink.save2new	WeblinksControllerWeblink	save()
Save as Copy (Сохранить в виде копии)	weblink.save2copy	WeblinksControllerWeblink	save()
Close (Закрыть)	weblink.cancel	WeblinksControllerWeblink	cancel()

Краткий обзор контроллеров в компонентах

Подведем краткий итог тому, что было ранее сказано о назначении контроллеров. В точке входа в компонент по шаблону MVC выполняются три команды.

- Получить контроллер для данной задачи.
- Выполнить задачу, используя контроллер.
- Произвести переадресацию, если таковая имеется.

Класс контроллер обозначается по имени компонента и названию задачи. Если задача отличается от простейшего отображения, то она выполняется, после чего устанавливается и производится переадресация на нужную страницу. А если задача не указана, то по умолчанию она считается задачей отображения и при ее выполнении отображается страница без переадресации.

Задачи сопоставляются с методами в контроллере. Для выполнения задачи из контроллера вызывается сопоставляемый с ней метод. С одной стороны, имя метода может совпадать с названием задачи. А с другой стороны, имя метода можно сопоставить с задачей, используя метод `registerTask()`.

Модели компонента Weblinks

А теперь, когда стал понятнее принцип действия контроллеров, перейдем к рассмотрению моделей компонента Weblinks. Для моделей этого компонента имеются два основных класса: `WeblinksModelWeblink` из файла `models/weblink.php` и `WeblinksModelWeblinks` из файла `models/weblinks.php`. Эти классы обозначаются таким же образом, как и классы контроллеров. В классе `WeblinksModelWeblink` предоставляются методы, поддерживающие выполнение контроллерами задач, назначаемых для отдельных элементов веб-ссылок, а в классе `WeblinksModelWeblinks` — методы для отображения списка веб-ссылок в экранной форме Weblinks Manager: Weblinks.

Рассмотрим сначала, каким образом выполнение контроллерами задач поддерживается в классе `WeblinksModelWeblink`. Прежде всего необходимо выбрать подходящий класс модели, а для этого нужно знать, каким образом контроллеры связываются с моделями. Напомним, что в классе `WeblinksControllerWeblinks` переопределяется метод `getModel()` для загрузки явным образом модели из класса `WeblinksModelWeblink`. Именно таким образом этот контроллер связывается с данной моделью. А в классе `WeblinksControllerWeblink` метод `getModel()` отсутствует, поэтому для получения модели в данном классе используется метод `getModel()` из его родительского класса `JControllerForm`. Ниже приведен исходный код этого метода.

```
public function getModel($name = '', $prefix = '',
↳$config = array('ignore_request' => true))
{
  if (empty($name)) {
    $name = $this->context;
  }

  return parent::getModel($name, $prefix, $config);
}
```

В этом методе переменной `$name` присваивается содержимое поля `context` объекта текущего класса. Конструктором данного класса в поле `context` устанавливается слово,

следующее после слова `Controller` в имени класса. В данном случае это слово `Weblink`, поскольку класс называется `WeblinksControllerWeblink`. Таким образом, при выполнении метода `getModel()` из родительского класса `JController` получается первая часть имени модели. Если переменная `$prefix` пуста, как в данном случае, то используется поле `model_prefix`. Это поле устанавливается двумя способами: из элемента массива в аргументе `$config` или из первой части имени класса.

В данном случае поле `model_prefix` не указано в переменной `$config`, и поэтому используются обычные условные обозначения имен. В конечном итоге получается исходное имя модели, похожее на имя контроллера, только в нем вместо слова `Controller` присутствует слово `Model`. Но такие условные обозначения имен можно легко изменить, указав значение поля `model_prefix` в переменной `$config` при получении экземпляра (т.е. создании) модели.

Рассмотрим далее, каким образом в модели поддерживаются задачи, инициируемые контроллерами. В табл. 7.4 и 7.5 перечислены методы контроллеров, вызываемые для каждой задачи, которую пользователь может выполнять в компоненте Weblinks. В частности, методы `add()`, `edit()` и `cancel()` вызываются из класса `WeblinksControllerWeblink` при загрузке или закрытии экранной формы редактирования веб-ссылки. Аналогичные методы в модели отсутствуют.

В методе `add()` просто проверяется, разрешено ли пользователю добавлять элемент веб-ссылки. Если это ему разрешено, то устанавливается и производится переадресация для открытия экранной формы редактирования новой веб-ссылки. И все это делается без доступа к методам модели.

В методе `add()` также проверяются полномочия пользователя. А кроме того, в нем из модели вызываются методы `getTable()`, `checkout()` и `getError()` с целью проверить, не снят ли данный элемент с регистрации, а затем пометить этот элемент как снятый с регистрации, т.е. заблокированный, прежде чем начать сеанс его редактирования. После этого устанавливается и производится переадресация для открытия экранной формы редактирования выбранного элемента.

В методе `cancel()` из модели вызывается метод `checkin()` для регистрации элемента, после чего производится переадресация обратно к экранной форме Weblinks Manager: Weblinks.

В остальных методах, в том числе `publish()`, `checkin()`, `delete()`, `reorder()`, `save()` и `saveorder()`, вызываются соответствующие методы из модели. На рис. 7.7 приведены методы, доступные в классе `WeblinksModelWeblink` и наследуемые из класса `JModelAdmin`. Как и следовало ожидать, в модели присутствуют те же самые шесть открытых методов (на рис. 7.7 они обозначены кружком).

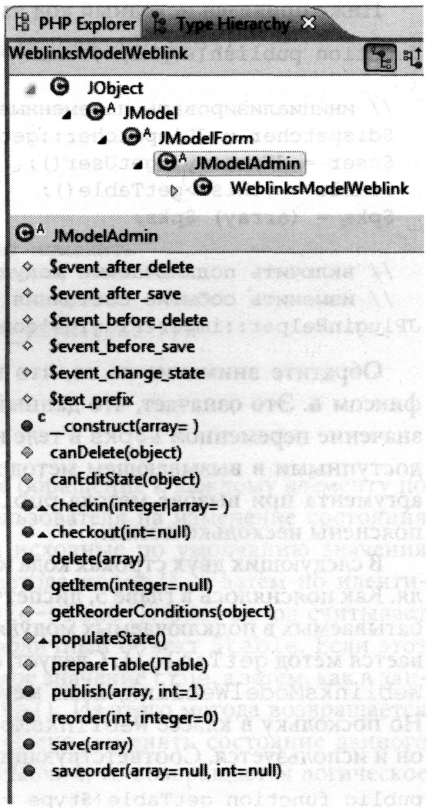


Рис. 7.7. Методы из класса `JModelAdmin`

Метод publish() из модели

Ранее был рассмотрен метод publish(), наследуемый классом WeblinksControllerWeblinks из класса JControllerAdmin. В этом методе был проанализирован следующий фрагмент кода:

```
// опубликовать элементы
    if (!$model->publish($cid, $value)) {
        JError::raiseWarning(500, $model->getError());
}
```

Именно здесь из модели вызывается метод publish(). Если он завершается успешно, то возвращается логическое значение true. В противном случае возвращается логическое значение false и предоставляются сведения о возникшей ошибке, которые могут быть извлечены методом getError().

А теперь рассмотрим аналогичный метод publish() из модели, а по существу, из класса JModelAdmin. Он вызывается с двумя аргументами. Напомним, что первым его аргументом является переменная $cid, содержащая массив идентификаторов веб-ссылок, проверенных по списку, а вторым аргументом — переменная $value, содержащая значение, которое требуется поместить в столбце state таблицы #__weblinks.

Ниже приведен исходный код из первой части метода publish().

```
function publish(&$pks, $value = 1)
{
  // инициализировать переменные
  $dispatcher = JDispatcher::getInstance();
  $user = JFactory::getUser();
  $table = $this->getTable();
  $pks = (array) $pks;

  // включить подключаемые модули содержимого, чтобы
  // изменить событие состояния
JPluginHelper::importPlugin('content');
```

Обратите внимание на то, что первый аргумент данного метода обозначается префиксом &. Это означает, что данный аргумент передается по ссылке. Так, если изменить значение переменной &$pks в теле вызываемого метода, внесенные изменения окажутся доступными в вызывающем методе из переменной $cid, использовавшейся в качестве аргумента при вызове метода publish(). Причины такого проектного решения будут пояснены несколько позже.

В следующих двух строках кода инициализируются объекты диспетчера и пользователя. Как пояснялось в главе 5, диспетчер используется для инициирования событий, обрабатываемых в подключаемых модулях. Далее создается табличный объект, для чего вызывается метод getTable(). Следует, однако, иметь в виду, что все это происходит в классе WeblinksModelWeblink, хотя метод publish() вызывался из класса JModelAdmin. Но поскольку в классе WeblinksModelWeblink имеется метод getTable(), то именно он и используется. Соответствующий код довольно прост:

```
public function getTable($type = 'Weblink', $prefix = 'WeblinksTable',
↳$config = array())
{
```

```
  return JTable::getInstance($type, $prefix, $config);
}
```

В этом коде объект класса WeblinksModelWeblink получается из папки tables. В этом классе предоставляются методы, используемые для чтения и записи информации в созданную таким образом таблицу базы данных.

В следующей строке кода переменная $pks преобразуется в массив. В данном случае она уже является массивом, и поэтому в этой строке кода никаких особых изменений не происходит. Но это дает возможность вызывать метод publish() как с одним целочисленным элементом, так и с целым массивом. Так, если вызвать его с одним идентификатором, последний будет преобразован в массив.

Далее следуют комментарий и последняя строка из первой части метода publish():

```
// включить подключаемые модули содержимого, чтобы
// изменить событие состояния
JPluginHelper::importPlugin('content');
```

В этой строке кода импортируются подключаемые модули содержимого, которые выполняются при инициировании соответствующих событий. В частности, при изменении состояния "опубликовано" конкретного элемента наступает событие onContent ChangeState.

Ниже приведен исходный код из следующей части метода publish().

```
// Проверка прав доступа.
foreach ($pks as $i => $pk) {
  $table->reset();

  if ($table->load($pk)) {
    if (!$this->canEditState($table)) {
      // избавиться от элементов, которые нельзя изменить
      unset($pks[$i]);
      JError::raiseWarning(403,
↳JText::_('JLIB_APPLICATION_ERROR_EDITSTATE_NOT_PERMITTED'));
      return false;
    }
  }
}
```

В этом фрагменте кода организуется циклическое обращение к каждому элементу по его идентификатору и проверяются полномочия пользователя на изменение состояния "опубликовано" каждого такого элемента. Сначала исходные по умолчанию значения устанавливаются в столбцах таблицы с помощью метода reset(), а затем по идентификатору каждого элемента вызывается метод $table->load(). Этот метод считывает строку из таблицы базы данных и загружает ее в табличный объект $table. Если этот метод завершится успешно, то возвращается логическое значение true, а затем, как в данном случае, вызывается метод $this->canEditState(). Из этого метода возвращается логическое значение true, если пользователю разрешено изменять состояние данного элемента веб-ссылки. А если у него нет таких полономочий, то возвращается логическое значение false.

Таким образом, если пользователь наделен соответствующими полномочиями, следующий далее кодовый блок условного оператора if пропускается и выполнение

рассматриваемого здесь метода продолжается дальше. А если у пользователя нет таких полномочий, то выполняются следующие действия. Во-первых, элемент удаляется по своему идентификатору из массива `$pks` с помощью команды `unset` языка PHP. Во-вторых, выводится сообщение, предупреждающее пользователя о том, что у него нет надлежащих полномочий. И в-третьих, происходит возврат из рассматриваемого здесь метода по команде `return false`. Следует заметить, что где бы команда `return` ни использовалась в теле вызываемого метода, его выполнение немедленно завершается и происходит возврат в вызывающий метод.

Напомним, что при передаче в качестве первого аргумента методу `publish()` переменная `$pks` была обозначена префиксом **&**, т.е. она передавалась по ссылке. Следовательно, при удалении элемента из массива, хранящегося в этой переменной, по команде `unset` изменения в массиве окажутся доступными в вызывающем методе. Иными словами, изменение переменной `$pks` в теле метода `publish()` отразится на состоянии переменной `$cid` в том контроллере, где этот метод вызывался.

Напомним также, что метод `publish()` вызывался из модели в теле метода `publish()` из класса `JControllerAdmin` следующим образом:

```
if (!$model->publish($cid, $value)) {
```

а затем количество опубликованных веб-ссылок было включено в сообщение так, как показано ниже.

```
$this->setMessage(JText::plural($ntext, count($cid)));
```

Если пользователь попытается удалить в корзину три веб-ссылки, но одна из них относится к той категории, где у него нет полномочий выполнять действие `core.edit.state`, то в конечном итоге в корзину попадут лишь две веб-ссылки. Но поскольку необработанный элемент все равно удаляется по команде `unset` из массива `$pks`, передаваемого данному методу по ссылке **&**, то изменения в массиве `$pks` отразятся на состоянии переменной `$cid` в вызывающем методе `publish()` текущего контроллера. Таким образом, переменная `$cid` в данном примере будет содержать лишь два элемента, а в сообщении будет правильно извещено об удалении в корзину двух, а не трех веб-ссылок.

Рассмотрим далее метод `canEditState()` из класса `WeblinksModelWeblink`. Ниже приведен его исходный код.

```
protected function canEditState($record)
    {
    $user = JFactory::getUser();

    if (!empty($record->catid)) {
      return $user->authorise('core.edit.state',
↳'com_weblinks.category.'.(int) $record->catid);
    }
    else {
      return parent::canEditState($record);
    }
}
```

Обратите внимание на то, что этот метод объявляется с модификатором доступа `protected`, а следовательно, он может использоваться только в текущем классе или

расширяющем его классе. Это означает, что данный метод предназначен для поддержки метода публикации, но не может быть использован за пределами своего класса.

В систему безопасности ACL (списков контроля доступа), внедренную в версии Joomla 1.6, введены специальные полномочия на изменение состояния "опубликовано" конкретного элемента. Для компонента Weblinks эти полномочия могут быть установлены как на уровне самого компонента, так и на уровне его категории. Поэтому в методе `canEditState()` проверяется, имеется ли категория у текущей записи. (Впрочем, у компонента Weblinks всегда должна быть какая-то категория, а следовательно, подобная проверка оказывается ненужной.) И если такая категория имеется, то вызывается метод `authorise()` из класса `JUser` для проверки полномочий текущего пользователя на изменение состояния "опубликовано" веб-ссылки в данной категории. (Обратите внимание на британский вариант написания метода `authorise()`, поскольку британский английский принят в качестве официального языка для Joomla.) В качестве аргументов метода `authorise()` указываются выполняемое действие и наименование ресурса. Для категорий наименование ресурса составляется по следующему образцу:

```
<имя компонента> + .category. + <идентификатор категории>
```

Если у текущей записи отсутствует категория, из родительского класса `JModelAdmin` вызывается метод `canEditState()`, который содержит следующую строку кода:

```
return $user->authorise('core.edit.state', $this->option);
```

В этой строке кода проверяются полномочия на уровне компонента. Поле `$this->option` содержит имя компонента (в данном случае `com_weblinks`).

Ранее в этой главе упоминался файл `access.xml`, который содержит действия ACL, выполняемые на уровне компонента и на уровне категории данного компонента. Кроме того, в этом файле определяются элементы разметки `action` с атрибутом `name="core.edit.state"` для обозначения компонентов и категорий по частям. Файл `access.xml` позволяет отредактировать установленные полномочия для компонентов и категорий. А в коде рассматриваемого здесь метода эти полномочия проверяются.

Если пользователю разрешено выполнить указанное действие, из метода `authorise()` возвращается логическое значение `true`, в противном случае — логическое значение `false`. Полученное в итоге значение возвращается далее из метода `canEditState()`.

Ниже приведен следующий фрагмент кода из метода `publish()` класса `JModelAdmin`.

```
// попытаться изменить состояние записей
if (!$table->publish($pks, $value, $user->get('id'))) {
    $this->setError($table->getError());
    return false;
}
```

В этом фрагменте кода, по существу, изменяется значение в базе данных. Для этого вызывается метод `publish()` из табличного объекта `$table`, возвращающий логическое значение `true` или `false` для обозначения своего успешного или неудачного завершения соответственно. Если этот метод завершится неудачно, из него будет выведено сообщение об ошибке и возвращено логическое значение `false`, указывающее вызывающему методу на возникшую ошибку.

В следующем фрагменте кода из метода publish(), определяемого в классе JModelAdmin, вызываются подключаемые модули для обработки события onContentChangeState, как показано ниже.

```
$context = $this->option . '.' . $this->name;

// инициировать событие onContentChangeState
$result = $dispatcher->trigger($this->event_change_state,
↳array($context, $pks, $value));

if (in_array(false, $result, true)) {
    $this->setError($table->getError());
    return false;
}
```

Сначала в этом фрагменте кода задается контекст, чтобы подключаемые модули могли проверить его по мере надобности. Затем созданный ранее объект $dispatcher используется для инициирования упомянутого выше события. В поле $this->event_change_state конструктором класса JModelAdmin было установлено исходное по умолчанию значение onContentChangeState.

Метод trigger() возвращает массив результатов обработки в каждом подключаемом модуле. Если обработка в любом из подключаемых модулей завершится неудачно, из таблицы будет снова выдано сообщение об ошибке и возвращено логическое значение false. Это очень важное свойство данной модели. Ведь оно позволяет разработчику специально настроить последовательность операций для публикации элементов. Так, если написать специальный подключаемый модуль для обработки события onContentChangeState, то в нем можно предусмотреть контроль над успешным или неудачным выполнением задачи публикации. В частности, можно потребовать от администратора веб-сайта дать согласие на публикацию пользователем конкретного элемента содержимого. И такой подключаемый модуль может управлять этим процессом.

Ниже приведен завершающий фрагмент кода из метода publish(), определяемого в классе JModelAdmin.

```
// очистить кеш компонента
$this->cleanCache();

$return true;
```

Метод cleanCache() очищает кеш-файлы компонента Weblinks. Если кеширование активизировано в Joomla, веб-страницы временно сохраняются в кеш-файлах. Поэтому если пользователь запросит ту же самую, отображавшуюся ранее веб-страницу, то ее получение можно ускорить, используя сохраненный кеш-файл. Но когда данные изменяются в таблице, любые кешированные файлы, содержащие данные из этой таблицы, могут оказаться неактуальными и устаревшими. Таким образом, эти файлы необходимо вовремя очистить, чтобы исключить риск отображения пользователю устаревшей информации.

И в последней строке кода из рассматриваемого здесь метода возвращается логическое значение true, обозначающее его успешное завершение. Если выполнение данного метода дошло до этой стадии, значит, можно с уверенностью утверждать, что никаких серьезных ошибок не произошло.

Метод save() из модели

Метод save() из класса JModelForm высвечивает ряд важных аспектов сохранения информации в базе данных. Когда элемент веб-ссылки сохраняется после ввода или редактирования, вызывается метод save() из класса JControllerForm, являющегося родительским для класса WeblinksControllerWeblink. Ниже приведен исходный код этого метода.

```
// проверить отправленные данные на достоверность
// Иногда в форме требуется вводить некоторые отправляемые данные,
// предназначенные, например, для модулей и подключаемых модулей.
$form = $model->getForm($data, false);
if (!$form) {
    $app->enqueueMessage($model->getError(), 'error');
    return false;
}
// проверить достоверность данных
$validData = $model->validate($form, $data);
```

В этом методе получается экземпляр объекта класса JForm с помощью метода getForm(), более подробно рассматриваемого в главе 8. В последней, выделенной полужирным строке кода из модели вызывается метод validate(). Этот метод находится в классе JModelForm. Напомним, что рассматриваемая здесь модель реализована в классе WeblinksModelWeblink, расширяющем класс JModelAdmin, который, в свою очередь, расширяет класс JModelForm. А поскольку в этом и родительском классах отсутствует метод validate(), то он вызывается из класса JModelForm, следующего вверх по иерархии наследования.

Метод validate() содержит следующий фрагмент кода:

```
$data = $form->filter($data);
$return = $form->validate($data);
```

Рассмотрим этот фрагмент кода более подробно, поскольку он очень важен для понимания рассматриваемого здесь процесса сохранения данных. В первой строке этого фрагмента кода вызывается метод filter() из класса JForm. Фильтрация данных служит для защиты базы данных от записи в нее зловредной информации, например, со стороны злоумышленника. В качестве примера такой фильтрации можно привести требование, чтобы адрес электронной почты содержал только строковые символы.

Во второй строке кода вызывается метод validate() из класса JForm. Проверка достоверности позволяет выяснить, насколько данные в поле значимы для этого поля. Например, для проверки достоверности адреса электронной почты необходимо убедиться в том, что в нем после ряда символов следует знак @, а за ним — имя домена.

Нередко при проверке достоверности данных пользователь уведомляется о необходимости ввести данные еще раз. С другой стороны, фильтрация данных может происходить незаметно для пользователя, удаляя зловредные данные перед их сохранением.

Другие методы модели

Сделаем беглый обзор остальных методов, поддерживающих выполнение задач в контроллере и относящихся к классу WeblinksModelWeblink, наследующему от класса JModelAdmin. В частности, методы save(), delete(), reorder(), saveorder() и

checkin() следуют тому же самому образцу, который был рассмотрен ранее на примере метода publish(). В каждом случае метод контроллера делает следующее.

- Выполняет некоторую подготовительную работу.
- Получает экземпляр модели.
- Вызывает метод модели с тем же самым именем.
- Сохраняет сообщение об успешном или неудачном завершении вызываемого метода.
- Возвращает логическое значение true при своем успешном завершении, а иначе — логическое значение false.

Подробное рассмотрение всех методов модели (даже для одного компонента Weblinks) выходит за рамки этой книги. Тем не менее, зная, каким образом они взаимодействуют друг с другом, вы можете проследить процесс обработки данных в каждом из них и понять их назначение и функции.

Табличный класс компонента Weblinks

При более углубленном рассмотрении методов модели можно обнаружить, что они, в свою очередь, вызывают методы из табличного класса. Так, в методе save() из класса WeblinksModelWeblink вызываются методы load(), bind(), check() и store() из класса WeblinksTableWeblink. Именно в этих методах и происходит фактическое взаимодействие с базой данных. Более подробно они описываются в последующих подразделах.

Табличный метод load()

Этот метод наследуется из класса JTable и загружает данные из строки базы данных, привязывая столбцы ее таблицы к полям экземпляра объекта типа JTable. Строка загружается из таблицы по первичному ключу последней (в данном примере — идентификатору столбца в таблице #__weblinks).

Самое замечательное, что вся необходимая для этого предварительная работа уже проделана в Joomla, а именно в родительском классе. А классу WeblinksTableWeblink остается только унаследовать ее результаты, ничего не меняя.

Табличный метод bind()

Этот метод принимает в качестве аргумента ассоциативный массив или объект и загружает каждый его элемент в поле объекта типа JTable. Например, в результате считывания строки из таблицы #__weblinks образуется ассоциативный массив в следующей строке кода из метода JTable::load():

```
$row = $this->_db->loadAssoc();
```

Этот массив содержит ключ к каждому столбцу таблицы со значением, равным значению из загруженной строки таблицы. Ниже приведена последняя строка из рассматриваемого здесь метода.

```
return $this->bind($row);
```

Метод bind() загружает все значения из массива в поля объекта типа JTable. Для каждого столбца таблицы в классе JTable имеется отдельное поле. Эти поля создаются в конструкторе класса JTable. А поскольку класс WeblinksTableWeblink расширяет класс JTable, то остается лишь запрограммировать те действия, которые являются особенными для данной таблицы. Именно это и делается в методе bind().

Особая обработка, которую приходится выполнять в классе WeblinksTableWeblink, относится к столбцам params и metadata таблицы #__weblinks. Если в таблице имеется однозначное соответствие ее столбцов полям из класса JTable, то для такой обработки достаточно воспользоваться методом bind() из этого класса. Но в столбцах params и metadata данные сохраняются в виде символьных строк формата JSON. Напомним, что символьные строки формата JSON позволяют сохранить в одном столбце базы данных несколько параметров. Это означает, что параметры можно добавлять и удалять, не изменяя структуру таблицы в базе данных. Хотя это и требует некоторых затрат: прежде чем воспользоваться значениями в коде, их придется распаковать в более пригодный формат.

Распаковка происходит в методе getItem() из класса WeblinksModelWeblink. Сначала в этом методе вызывается метод getItem() из родительского класса JModelAdmin, преобразующий символьную строку формата JSON из столбца params в массив с помощью метода toArray() из класса JRegistry. А после возврата в метод getItem() из класса WeblinksModelWeblink то же самое происходит с данными из столбца metadata. Это позволяет обрабатывать отдельные поля с символьными строками формата JSON как элементы массива средствами класса WeblinksTableWeblink.

Когда же наступает момент записи строки назад в таблицу #__weblinks, приходится выполнять обратную операцию, преобразуя элементы массива в символьные строки формата JSON. И это делается в приведенном ниже фрагменте кода из метода bind(), вызываемого из класса WeblinksTableWeblink.

```
public function bind($array, $ignore = '')
{
  if (isset($array['params']) && is_array($array['params'])) {
    $registry = new JRegistry();
    $registry->loadArray($array['params']);
    $array['params'] = (string)$registry;
  }

  if (isset($array['metadata']) && is_array($array['metadata'])) {
    $registry = new JRegistry();
    $registry->loadArray($array['metadata']);
    $array['metadata'] = (string)$registry;
  }

  return parent::bind($array, $ignore);
}
```

В этом методе переменная $array содержит ассоциативный массив полей данных для таблицы #__weblinks. В качестве ключа к этому массиву служит имя столбца, а в качестве значения — конкретное значение, записываемое обратно в базу данных. В коде данного метода проверяется наличие массива в элементе массива $array['params']. Если такой массив имеется, он преобразуется сначала в объект типа JRegistry, а затем

обратно в символьную строку формата JSON с помощью метода `__toString()` из класса `JRegistry`. Напомним, что для приведения объекта к типу символьной строки с помощью оператора `(string)` для данного объекта вызывается метод `__toString()`. Та же самая последовательность действий повторяется для преобразования метаданных из массива в символьную строку формата JSON. И в последней строке кода из рассматриваемого здесь метода вызывается метод `bind()` из родительского класса, что очень важно, поскольку именно здесь выполняется остальная часть обработки, относящаяся к привязке других столбцов к табличному объекту.

Табличный метод `store()`

Метод `store()` действует по тому же самому образцу. Сначала выполняется обработка данных, характерная для данной таблицы, а затем вызывается метод `store()` из родительского класса для выполнения стандартной обработки.

Прежде всего в рассматриваемом здесь методе `store()` выясняется, что именно нужно делать: редактировать уже имеющуюся строку таблицы или же создавать новую. Исходя из этого обновляются столбцы видоизмененной даты (`modifed date`) и видоизмененной пользователем даты (`modified_by user`) или же столбцы созданной даты (`created date`) и созданной пользователем даты (`created_by user`). Затем выполняется специальная обработка с целью убедиться в том, что элемент с тем же самым псевдонимом и идентификатором категории отсутствует. С помощью псевдонима и категории формируется URL для веб-ссылок. При наличии дубликатов формируемый URL окажется неоднозначным, и поэтому он может указывать не на тот элемент. Именно это условие и проверяется при сохранении элемента, а при обнаружении ошибки выводится соответствующее сообщение. На этом специальная обработка, необходимая для веб-ссылок, завершается и далее вызывается метод `store()` из родительского класса, возвращающий результат своего выполнения.

Табличный метод `check()`

В методе `check()` проверяется достоверность или значимость данных в строке таблицы. Этот метод действует немного иначе, чем остальные табличные методы. Во-первых, в родительском классе `JTable` метод `check()` объявлен как пустой, и поэтому он не вызывается. И во-вторых, в этом методе предпринимается попытка исправить ошибку в данных, если такое вообще возможно, вместо того, чтобы просто сообщать о ней.

Для веб-ссылок в данном методе проверяется следующее.

- Содержит ли URL только достоверные символы.
- Не является ли заглавие пустым.
- Начинается ли URL с правильных символов (если требуется, добавляется префикс `http://`).
- Имеется ли уже веб-ссылка с таким же именем.
- Задан ли достоверный псевдоним (если он пустой, то задается).
- Правильно ли отформатированы ключевые слова, если они введены, и не содержат ли они запрещенные символы.

Если сравнить методы `check()` из разных базовых табличных классов, в том числе `WeblinksTableWeblink`, `ContactsTableContact`, `BannersTableBanner` и `NewsfeedsTableNewsfeed`, то в них можно обнаружить немало дублирующегося кода. В частности, каждый из них содержит один и тот же код для проверки и исправления дат публикации. Это положение можно было бы исправить, перенеся дублирующийся код в родительский класс и вызвав его метод `check()` для проверки из метода `check()` в каждом его подклассе, как это делается, например, в методе `store()` или `publish()`. Следует, однако, иметь в виду, что удаление дублирующегося кода никак не изменило бы функционирование самих программ, а только сократило бы размеры кодовой базы. Ведь дублирующийся код затрудняет сопровождение программ. Так, если потребуется исправить программные ошибки в дублирующемся коде, это придется сделать в нескольких местах программы.

Процесс усовершенствования кода без изменения его функционирования называется реорганизацией кода. Такая реорганизация может быть произведена по целому ряду причин, включая и сокращение дублирующегося кода, чтобы упростить код, облегчить его чтение и улучшить его стандартизацию. В такой крупной и сложной программе, как Joomla, реорганизация кода происходит постоянно. И поэтому очень важно, чтобы она не приводила к случайным изменениям в работе программы и не нарушала ее функционирование.

Резюме

В этой главе рассматривалось применение контроллеров и моделей по шаблону проектирования MVC на примере административной части компонента Weblinks. В ней также был рассмотрен класс `JTable`, помогающий организовать взаимодействие с базой данных.

А в следующей главе знакомство с шаблоном MVC будет продолжено рассмотрением представлений на том же самом примере административной части компонента Weblinks. Затем на примере пользовательской части компонента Weblinks будет показано, каким образом компоненты этой части веб-сайта организуются и действуют по шаблону MVC.

8

Компоненты, часть II. Представления, класс JForm и пользовательская часть

В предыдущей главе были рассмотрены контроллеры и модели по шаблону MVC на примере административной части компонента Weblinks. А в этой главе рассмотрение данного компонента будет продолжено в связи с представлениями.

Сначала будут исследованы представления и методы отображения вместе с более углубленным анализом компоновки компонента Weblinks Manager. Затем будет рассмотрено применение класса `JForm` в представлении Weblink Edit (Правка веб-ссылок). И в завершение главы будет описана пользовательская часть компонента Weblinks, а также обсуждены сходства и отличия компонентов административной и пользовательской части веб-сайта.

Представления и методы отображения

Итак, перейдем к исследованию представительской составляющей шаблона MVC, не покидая административную часть веб-сайта. Напомним, что представление отображается в отсутствие конкретной задачи или в том случае, если это задача отображения. В таком случае класс `WeblinksController` загружается из файла `controller.php` относительно исходной папки компонента `administrator/components/com_weblinks/`. (Имена всех файлов обозначаются в этой главе относительно данной папки, если не указано иное.) Единственным в этом классе является метод `display()`, отвечающий за отображение двух представлений компонента. Так, в выбираемом по умолчанию представлении отображается список веб-ссылок (`Weblinks`). А кроме того, имеется представление, в котором разрешается вводить и редактировать одиночную веб-ссылку (`Weblink`).

Представление списка веб-ссылок

Начнем наше исследование с представления списка веб-ссылок. Ниже приведен исходный код метода `display()` из класса `WeblinksController`.

```
public function display($cachable = false, $urlparams = false)
{
  require_once JPATH_COMPONENT.'/helpers/weblinks.php';

  // загрузить подменю
  WeblinksHelper::addSubmenu(JRequest::getCmd('view', 'weblinks'));

  $view = JRequest::getCmd('view', 'weblinks');
  $layout = JRequest::getCmd('layout', 'default');
  $id = JRequest::getInt('id');

  // проверить наличие формы редактирования
  if ($view == 'weblink' && $layout == 'edit' &&
↳!$this->checkEditId('com_weblinks.edit.weblink', $id)) {

    // не разрешать несанкционированный доступ к форме
    $this->setError(JText::sprintf('JLIB_APPLICATION_ERROR_UNHELD_ID',
↳$id));
    $this->setMessage($this->getError(), 'error');
    $this->setRedirect(
↳JRoute::_('index.php?option=com_weblinks&view=weblinks', false));

    return false;
  }

  parent::display();

  return $this;
}
```

Этот метод получает представление, компоновку и идентификатор элемента веб-ссылок из запроса. В данном случае имена представления и компоновки оказываются пустыми, и поэтому по умолчанию выбираются имена `weblinks` для представления и `default` для компоновки. Идентификатор также оказывается пустым.

Следующий кодовый блок пропускается, поскольку это не представление одиночных веб-ссылок. Затем выполняется метод `display()` из родительского класса. И наконец, возвращается текущий объект. Следует, однако, иметь в виду, что возвращаемое значение не используется, поскольку рассматриваемый здесь метод вызывается без присваивания переменной результата его выполнения. Но в то же время возврат объекта позволяет связывать методы в цепочку. Благодаря этому объект, созданный одним методом, может быть использован для вызова другого метода. Например, три следующие строки кода:

```
$controller = JController::getInstance('Weblinks');
$controller->execute(JRequest::getCmd('task'));
$controller->redirect();
```

можно заменить приведенной ниже единственной строкой кода.

```
JController::getInstance('Weblinks')->
↳execute(JRequest::getCmd('task')->redirect();
```

Дело в том, что метод `getInstance()` возвращает объект контроллера, а у этого объекта имеется метод `execute()`, который, в свою очередь, возвращает объект контроллера. И этим объектом можно воспользоваться для вызова метода `redirect()`. Возможность связывать методы в цепочку было внедрена в версии 5 языка PHP.

Метод `display()` из класса `JController`

Класс `WeblinksController` расширяет класс `JController` из файла `libraries/joomla/application/component/controller.php`, поэтому по ссылке `parent::display()` в классе `WeblinksController` вызывается на выполнение метод `display()` из класса `JController`. Ниже приведена первая часть исходного кода этого метода.

```
public function display($cachable = false, $urlparams = false)
{
  $document = JFactory::getDocument();
  $viewType = $document->getType();
  $viewName = JRequest::getCmd('view', $this->default_view);
  $viewLayout = JRequest::getCmd('layout', 'default');

  $view = $this->getView($viewName, $viewType, '', array('base_path' =>
↳$this->basePath, 'layout' => $viewLayout));

  // получить/создать модель
  if ($model = $this->getModel($viewName)) {
    // сделать модель используемой в представлении по умолчанию
    $view->setModel($model, true);
  }

  $view->assignRef('document', $document);

  $conf = JFactory::getConfig();
```

В этой части исходного кода метода `display()` получается документ, которым в данном случае оказывается объект типа `JDocumentHTML`, тип представления — HTML, имя представления — `weblinks`, имя компоновки — `default`, класс представления — `WeblinksViewWeblinks`. Затем получается модель из класса `WeblinksModelWeblinks`, которая становится используемой по умолчанию в данном представлении. Далее задается компоновка для представления и в его классе создается поле `document`, которому присваивается объект документа (из переменной `$document`). И наконец, будут получены сведения о глобальной настройке данного представления, которые затем сохраняются в переменной `$conf`.

В следующей (не показанной здесь) части исходного кода метода `display()` проверяется, сохранен ли данный документ (т.е. веб-страница) в кеш-файле. И если он сохранен, то загружается из кеша. В противном случае вызывается метод `display()` из текущего представления (в данном случае из класса `WeblinksViewWeblinks`, находящегося в файле `views/weblinks/view.html.php`).

Метод display() из класса WeblinksViewWeblinks

Ниже приведен исходный код этого метода.

```
public function display($tpl = null)
{
  $this->state                = $this->get('State');
  $this->items                = $this->get('Items');
  $this->pagination           = $this->get('Pagination');

  // проверить ошибки
  if (count($errors = $this->get('Errors'))) {
    JError::raiseError(500, implode("\n", $errors));
    return false;
  }

  $this->addToolbar();
  parent::display($tpl);
}
```

В этом методе, по существу, начинается создание веб-страницы. В трех первых строках его кода из модели поочередно вызываются методы getState(), getItems() и getPagination(). В частности, из представления вызывается метод get() для получения модели и поиска в ней метода под именем, состоящим из префикса get и строки аргумента (например, get + State). Напомним, что класс модели WeblinksModelWeblinks расширяет класс JModelList, а тот, в свою очередь, — класс JModel.

В данном случае метод getState() наследуется из класса JModel, тогда как методы getItems() и getPagination() — из класса JModelList. Метод getState() возвращает объект состояния, который представляет текущее состояние экранной формы, включая действующие фильтры, номер страницы, порядок расположение элементов и пр. Метод getItems() возвращает перечень элементов веб-ссылок для отображения списком на экране. А метод getPagination() возвращает объект разбиения на страницы, служащий для указания номеров страниц на экране и поочередного перемещения по элементам на странице.

Получив из модели нужные сведения, можно проверить ошибки. Если таковые отсутствуют, вызывается метод addToolbar() для вывода на экран панели инструментов с пиктограммами New, Edit, Publish и т.д. Напомним, что панель инструментов берется из административного модуля mod_toolbar. Этот модуль служит для отображения глобального объекта панели инструментов. Метод addToolbar() создает пиктограммы панели инструментов в этом объекте, чтобы сделать их доступными для последующего отображения при воспроизведении модуля mod_toolbar в цикле выполнения команд.

Затем вызывается метод display() из родительского класса JView, находящегося в файле libraries/joomla/application/component/view.php. Ниже приведен исходный код этого метода.

```
function display($tpl = null)
{
  $result = $this->loadTemplate($tpl);
  if (JError::isError($result)) {
    return $result;
  }

  echo $result;
}
```

В этом методе, в свою очередь, вызывается метод `loadTemplate()` из класса `JView`, а результат его выполнения сохраняется в переменной `$result`. Затем выполняется команда `echo` языка PHP для отправки полученного результата браузеру. Как пояснялось в главе 3, когда выполняется компонент, выходной буфер уже активизирован с помощью функции `ob_start()` языка PHP. Поэтому когда выдается команда `echo`, результат выводится в выходной буфер, а не отправляется непосредственно в браузер.

В методе `loadTemplate()` проверяется наличие файлов замещения шаблонов и загружаются любые требующиеся файлы языковой поддержки. А в завершающей части данного метода выполняется приведенный ниже код, при условии, что обнаружен подходящий файл компоновки.

```
// начать вывод результатов в выходной буфер
ob_start();
// включить файл запрашиваемого шаблона в локальную область действия
// (выполнить для этого логику представления)
include $this->_template;

// завершить действия с запрашиваемым шаблоном;
// получить буфер и очистить его
$this->_output = ob_get_contents();
ob_end_clean();
return $this->_output;
```

Сначала функция `ob_start()` активизирует вывод в выходной буфер, а затем включается файл компоновки шаблона. В данном случае это будет файл `views/weblinks/tmpl/default.php`. Если бы вместо исходного шаблона Bluestork использовался административный шаблон Hathor, то пришлось бы загрузить файл `administrator/templates/hathor/html/com_weblinks/weblinks/default.php`, замещающий шаблон Hathor.

В данной компоновке класс не объявляется, и поэтому ее файл выполняется сразу, а результат, выводимый по команде `echo`, сохраняется в выходном буфере. Затем содержимое этого буфера копируется в поле `$this->_output`, возвращаемое вызывающему методу (в переменной `$result`, как следует из предыдущего кодового блока).

Файл исходной компоновки

А теперь перейдем к рассмотрению файла исходной компоновки, в котором фактически отображается результат выполнения компонента. В частности, покажем, что же отображается в каждой части компоновки представления. Для удобства рассмотрения этого файла на рис. 8.1 приведен моментальный снимок экрана Weblinks Manager: Weblinks, где метками **A**-**F** обозначены его отдельные части.

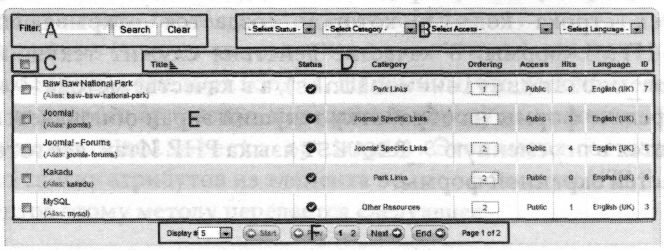


Рис. 8.1. Моментальный снимок экрана Weblinks Manager: Weblinks

Служебные операции

В первой части файла исходной компоновки выполняется ряд служебных операций, как показано ниже.

```
JHtml::addIncludePath(JPATH_COMPONENT.'/helpers/html');
JHtml::_('behavior.tooltip');
JHtml::_('script','system/multiselect.js', false, true);

$user = JFactory::getUser();
$userId = $user->get('id');
$listOrder = $this->escape($this->state->get('list.ordering'));
$listDirn = $this->escape($this->state->get('list.direction'));
$canOrder = $user->authorise('core.edit.state',
↳'com_weblinks.category');
$saveOrder = $listOrder == 'a.ordering';
```

В первой строке приведенного выше фрагмента кода указывается путь к файлу `admnistrator/com_weblinks/helpers/html`. Назначение этой строки имеет большое значение для разработки специальных компонентов. Она позволяет разработчикам включать собственные классы с префиксом `JHtml` и соответствующие методы, которые можно выполнять с помощью служебного метода `JHtml::_()`. Напомним, что этот метод автоматически загружает класс, если он еще не был загружен.

Для этой цели достаточно создать класс под названием `JHtml + <имя компонента>` (например, `JHtmlWeblinks`) и поместить его файл в папке `helpers` данного компонента. Так, если бы в классе `JHtmlWeblinks` был метод `myGrid()`, его можно было бы вызвать на выполнение по команде `JHtml::_('weblinks.mygrid', 'argument1', 'argument2')`, где `argument1` и `argument2` — первый и второй аргументы вызываемого метода.

Возвращаясь к рассматриваемой здесь исходной компоновке, следует заметить, что метод `JHtml::_('behavior.tooltip')` вызывается для добавления из библиотеки MooTools языка JavaScript кода, необходимого для отображения всплывающих подсказок, а также второго исходного файла сценария JavaScript под названием `multiselect.js`. Более подробно сценарии JavaScript рассматриваются в главе 12.

В следующих строках кода инициализируется ряд переменных, необходимых для компоновки, включая порядок составления списка и возможность изменения этого порядка текущим пользователем. Упомянутый выше сценарий загружается непосредственно оператором включения файлов, а следовательно, он по-прежнему остается в области действия класса `WeblinksViewWebinks`, в который и был включен. Таким образом, переменная `$this` ссылается на экземпляр (т.е. объект) данного класса.

Далее следует строка кода, в которой создается открывающий дескриптор `form` разметки HTML-формы. В качестве действия служит текущий URL (`index.php?option=com_weblinks&view=weblinks`), а в качестве метода — метод отправки. Поэтому при передаче формы на обработку текущий экран обновляется, а переменные формы помещаются в переменную `$_REQUEST` языка PHP. Итак, все готово к компоновке различных частей экранной формы.

Часть A. Фильтр заглавий

Ниже приведен код компоновки части A экранной формы, представленной на рис. 8.1.

```
<div class="filter-search fltlft">
    <label class="filter-search-lbl" for="filter_search"><?php echo
↳JText::_('JSEARCH_FILTER_LABEL'); ?></label>
    <input type="text" name="filter_search" id="filter_search"
↳value="<?php echo $this->escape($this->state->get('filter.search')); ?>"
↳title="<?php echo JText::_('COM_WEBLINKS_SEARCH_IN_TITLE'); ?>" />
    <button type="submit"><?php echo JText::_('JSEARCH_FILTER_SUBMIT');
↳?></button>
    <button type="button"
↳onclick="document.id('filter_search').value='';this.form.submit();">
↳<?php echo JText::_('JSEARCH_FILTER_CLEAR'); ?></button>
</div>
```

В этом коде создаются и размечаются четыре элемента из части A рассматриваемой здесь экранной формы: метка, поле ввода и две кнопки. Как и в других файлах компоновки, в данном случае для компоновки веб-страницы совместно используются элементы разметки HTML и PHP. Кнопка очистки (Clear) просто устанавливает пустое значение в поле ввода и передает форму для перезагрузки страницы.

Часть B. Списки выбора фильтров

В части B рассматриваемой здесь экранной формы содержатся четыре списка для выбора фильтров. Они дают пользователю возможность отфильтровать список веб-ссылок, исходя из состояния "опубликовано", категории, уровня доступа и выбранного языка. Ниже приведен код компоновки списка фильтров состояний.

```
    <select name="filter_published" class="inputbox"
↳onchange="this.form.submit()">
    <option value=""><?php echo
↳JText::_('JOPTION_SELECT_PUBLISHED');?></option>
    <?php echo JHtml::_('select.options',
↳JHtml::_('jgrid.publishedOptions'), 'value', 'text',
↳$this->state->get('filter.state'), true);?>
    </select>
```

В этом фрагменте кода создается элемент `select` разметки HTML-формы. Его первый атрибут `option` принимает пустое строковое значение `" "`, а также текст перевода метки на текущий язык по ключу `JOPTION_SELECT_PUBLISHED`. Затем в выделенной полужирным строке кода вызывается метод. Рассмотрим эту строку кода подробнее. В данном примере из метода `JHtml::_('select.options', . . .)` вызывается метод `JHtmlSelect::options()`, а из метода `JHtml::_('jgrid.publishedOptions')` — метод `JHtmlJGrid::publishedOptions()`. Напомним, что дополнительные аргументы, указываемые после первого аргумента, передаются методу как таковые. В данной строке кода вызывается метод `JHtmlSelect::options()`. Этот метод составляет список элементов выбора по списку атрибутов из элемента `select` разметки HTML-формы. В качестве пяти аргументов этому методу передается следующее.

- Массив значений. Устанавливается в результате выполнения метода `JHtmlJGrid::publishedOptions()` и представляет собой массив объектов для каждого возможного состояния "опубликовано" (**1** — "опубликовано", **0** — "снято с публикации", **2** — "архивировано", **-2** — "удалено в корзину" и ***** — все прочее).
- Имя переменной ссылки на объект. Устанавливается равным `"value"`.
- Имя текстовой метки варианта выбора. Устанавливается равным `"text"`.
- Выбранное значение. Устанавливается равным `$this->state->get('filter_state')` (т.е. выбранному в данный момент состоянию "опубликовано").
- Логическое значение, определяющее, следует ли переводить текст и метки. Устанавливается равным логическому значению `true`.

В результате выполнения этого фрагмента кода генерируется приведенный ниже исходный код элемента `select` для разметки списка, который можно увидеть при просмотре веб-страницы в исходном коде HTML.

```
    <select onchange="this.form.submit()" class="inputbox"
↳name="filter_published">
        <option value="">- Select Status - </option>
        <option value="1">Published</option>
        <option value="0">Unpublished</option>
        <option value="2">Archived</option>
        <option value="- 2">Trashed</option>
        <option value="*">All</option>
    </select>
```

Код разметки, выделенный выше полужирным, генерируется методом `JHtmlSelect::options()`. В атрибуте `onchange` установлено значение `"this.form.submit()"`, поэтому страница автоматически перезагружается всякий раз, когда это значение изменяется, причем для выбора отдельных элементов состояния используется новое значение фильтра.

Каждый фильтр устанавливается в форме одинаково, а для составления списка возможных вариантов выбора в соответствии с разметкой в элементе `select` используются разные методы из классов, производных от класса `JHtml`.

- Метод `JHtmlCategory::options()`. Составляет список категорий.
- Метод `JHtmlAccess::assetgroups()`. Составляет список уровней доступа.
- Метод `JHtmlContentLanguage::existing()`. Составляет список доступных языков.

Часть C. Флажок глобальной установки

Флажок глобальной установки компонуется в следующей строке кода:

```
<input type="checkbox" name="checkall-toggle" value=""
↳onclick="Joomla.checkAll(this)" />
```

При его установке вызывается метод `checkAll()` из сценария JavaScript, находящегося в файле `media/system/js/core.js`. В этом методе происходит циклическое обращение ко всем элементам флажков, которые устанавливаются в состояние, определяемое состоянием данного флажка.

Часть D. Сортируемые заголовки столбцов

В рассматриваемой здесь экранной форме имеются семь заголовков столбцов, которые можно отсортировать по каждому столбцу, щелкнув на нем. А если щелкнуть на столбце повторно, то столбцы будут отсортированы в обратном порядке. В приведенной ниже строке кода компонуется заголовок первого столбца с названием веб-ссылки.

```
<?php echo JHtml::_('grid.sort', 'JGLOBAL_TITLE', 'a.title', $listDirn,
↳$listOrder); ?>
```

В этой строке кода вызывается метод `JHtmlGrid::sort()` со следующим аргументами.

- Заглавие: `"JGLOBAL_TITLE"`.
- Поле упорядочения: `"a.title"` (столбец `title` из таблицы `#__weblinks`).
- Текущее направление сортировки: `$listDirn` (устанавливается в начале данного файла компоновки).
- Выбранное упорядочение: `$listOrder` (устанавливается в начале данного файла компоновки).

В этой единственной строке кода реализуется очень удобное для пользователя свойство: один и тот же метод используется для компоновки всех заголовков столбцов в экранной форме.

Часть E. Элементы веб-ссылок

Список элементов веб-ссылок выводится в элементе `tbody` разметки тела таблицы в коде HTML. Каждая веб-ссылка является строкой таблицы (элемент разметки `tr`), а каждый столбец — ячейкой таблицы (элемент разметки `td`). Как и следовало ожидать, строки таблицы выводятся в теле цикла `foreach` следующим образом:

```
<?php foreach ($this->items as $i => $item) :
```

Напомним, что в поле `$this->items` содержится массив элементов, возвращаемых моделью. Сначала над этим элементом производится ряд служебных операций: проверяется его упорядочение, получается ссылка на категорию, проверяются полномочия пользователя на создание новых элементов, редактируются уже имеющиеся элементы, регистрируются элементы и правится их состояние. Затем для данного элемента выводится каждый столбец таблицы.

При компоновке первого столбца таблицы вызывается метод `JHtmlGrid::id()` для вывода на экран флажка со значением идентификатора каждого элемента веб-ссылки. В этом методе компонуется флажок с именем `cid[]`. Квадратные скобки после имени флажка указывают на то, что результаты возвращаются из данного метода в виде массива идентификаторов для всех установленных флажков. Некоторые задачи, в том числе публикация, снятие с публикации, архивирование, регистрация, удаление в корзину и удаление, выполняются над элементами массива. При выполнении этих задач из элемента запроса `cid[]` считывается массив.

Следующий столбец таблицы компонуется сложнее. Если элемент снят с регистрации (т.е. заблокирован), пиктограмма замка выводится на экран в следующей строке кода:

```
<?php echo JHtml::_('jgrid.checkedout', $i, $item->editor,
↳$item->checked_out_time, 'weblinks.', $canCheckin); ?>
```

В этой строке кода вызывается метод `JHtmlJGrid::checkedout()` с индексом текущей строки таблицы, именем лица, снявшего элемент с регистрации, временем снятия с регистрации, префиксом задачи (в данном случае — `"weblinks"`) и логическим значением, обозначающим, разрешено ли пользователю зарегистрировать данный элемент. Напомним, что переменная `$canCheckin` была установлена в начале цикла `foreach` в зависимости от полномочий текущего пользователя на выполнение действия `core.manage` над компонентом `com_checkin`, а также от того, является ли он тем же самым пользователем, который снял данный элемент с регистрации. Если пользователю разрешено зарегистрировать элемент, на экран выводится пиктограмма в виде ссылки, инициирующей задачу регистрации элемента с данным идентификатором. В данном случае упомянутый выше метод генерирует следующий код разметки в формате HTML:

```
<a class="jgrid hasTip" href="#"
↳onclick="return listItemTask('cb0','weblinks.checkin')"
↳title="Checkin::Super User\&lt;br /&gt;Wednesday,
↳01 June 2011\&lt;br /&gt;18:15">
        <span class="state checkedout">
                <span class="text">Checked out
                </span>
        </span>
</a>
```

Пиктограмма, выбираемая из селектора классов CSS типа `jgrid span.checkedout`, представляет собой ссылку, по которой выполняется задача `weblinks.checkin` для регистрации элемента. Если же пользователю не разрешено зарегистрировать элемент, генерируется следующий код разметки в формате HTML:

```
<span class="jgrid hasTip"
↳title="Checked out::Super User\&lt;br /&gt;Wednesday,
↳01 June 2011\&lt;br /&gt;18:08">
        <span class="state checkedout">
            <span class="text">Checked out
            </span>
</span>
```

Этот код похож на предыдущий, за исключением того, что в нем отсутствует элемент привязки `a` со ссылкой на задачу. Тем не менее в том же самом столбце таблицы отображается заглавие веб-ссылки. Если пользователю разрешено редактировать веб-ссылку, ее заглавие будет отображено в виде ссылки на соответствующий элемент, а иначе — в текстовом виде. И наконец, отображается псевдоним элемента.

В столбце состояния указывается, был ли данный элемент опубликован. Для отображения соответствующей пиктограммы метод `JHtmlJGrid::published()` вызывается из следующей строки кода разметки:

```
<?php echo JHtml::_('jgrid.published', $item->state, $i, 'weblinks.',
↳$canChange, 'cb', $item->publish_up, $item->publish_down); ?>
```

В этой строке кода выводится пиктограмма, соответствующая текущему состоянию "опубликовано" данного элемента. Если текущий пользователь имеет полномочия на выполнение действия `edit.state`, а данный элемент находится в состоянии "опубликовано" или "снято с публикации", то появляется пиктограмма в виде ссылки, позволяющей

сменить состояние элемента с "опубликовано" на "снято с публикации", или наоборот. И все это делается в методе `JHtmlJGrid::published()`.

В следующем столбце таблицы просто отображается заглавие категории, как показано ниже.

```
<?php echo $this->escape($item->category_title); ?>
```



Следует заметить, что заглавие категории обрабатывается при вызове метода `$this->escape()`. Но в связи с тем, что заглавие категории вводится пользователем, оно может содержать злонамеренный код. В методе `escape()` удаляются любые символы, которые злоумышленник может ввести, пытаясь внедрить злонамеренный код JavaScript или SQL в компоновку веб-страницы. По умолчанию в подклассах, производных от класса `JView`, применяется в виде метода `escape()` функция `htmlspecialchars()`, встроенная в язык PHP. Но эту функцию можно заменить любым методом, соответственно установив поле `$_escape`, например, в конструкторе данного класса.

Вообще говоря, любые строковые данные, вводимые пользователем, рекомендуется экранировать. В данной форме экранируется значение, вводимое для поиска фильтра, заглавие и псевдоним веб-ссылки, заглавие категории, уровень доступа и название языка. Если же вводимое значение не содержит специальные символы, оно остается без изменения.

Далее следует столбец упорядочения. Его компоновка усложняется тем, что в нем поддерживается очень полезное свойство. Так, если список веб-ссылок упорядочен по категориям, то имеется возможность изменить упорядочение отдельных веб-ссылок. Это можно сделать одним из двух способов. Во-первых, для этой цели предоставляются стрелки перемещения веб-ссылки вверх или вниз по списку, а во-вторых, — поле, в котором можно ввести желаемый порядковый номер для одного или более элемента, а затем переупорядочить весь список. Дело усложняется еще и тем, что список веб-ссылок может быть отсортирован по восходящей или нисходящей. Так, если он отсортирован по восходящей, то назначение стрелок, направленных вверх и вниз, изменяется на противоположное.

Ниже приведен код компоновки данного столбца таблицы.

```
<?php if ($canChange) : ?>
    <?php if ($saveOrder) :?>
        <?php if ($listDirn == 'asc') : ?>
            <span><?php echo $this->pagination->orderUpIcon($i,
($item->catid == @$this->items[$i-1]->catid), 'weblinks.orderup',
'JLIB_HTML_MOVE_UP', $ordering); ?></span>
            <span><?php echo $this->pagination->orderDownIcon($i,
$this->pagination->total, ($item->catid == @$this->items[$i+1]->catid),
'weblinks.orderdown', 'JLIB_HTML_MOVE_DOWN', $ordering); ?></span>
        <?php elseif ($listDirn == 'desc') : ?>
            <span><?php echo $this->pagination->orderUpIcon($i,
 ($item->catid == @$this->items[$i-1]->catid), 'weblinks.orderdown',
'JLIB_HTML_MOVE_UP', $ordering); ?></span>
            <span><?php echo $this->pagination->orderDownIcon($i,
$this->pagination->total, ($item->catid == @$this->items[$i+1]->catid),
'weblinks.orderup', 'JLIB_HTML_MOVE_DOWN', $ordering); ?></span>
        <?php endif; ?>
    <?php endif; ?>
    <?php $disabled = $saveOrder ? '' : 'disabled="disabled"'; ?>
```

```
    <input type="text" name="order[]" size="5" value="<?php echo
↳$item->ordering;?>" <?php echo $disabled ?> class="text-area-order" />
<?php else : ?>
    <?php echo $item->ordering; ?>
<?php endif; ?>
```

Напомним, что в начале цикла `foreach` устанавливается переменная `$canChange`. Если она принимает логическое значение `true`, значит, текущий пользователь имеет полномочия на выполнение действия `edit.state` и регистрацию текущей веб-ссылки. Разрешение на регистрацию требуется для того, чтобы во время переупорядочения списка веб-ссылок можно было временно снять отдельные их элементы с регистрации (т.е. заблокировать), а по завершении этой операции — снова зарегистрировать (т.е. разблокировать).

В данном случае первый кодовый блок выполняется во внешнем условном операторе `if`. Затем проверяется, установлено ли в переменной `$saveOrder` логическое значение `true`. Напомним, что это значение было установлено в самом начале рассматриваемого здесь файла компоновки и является истинным, если сортировка происходит по столбцу упорядочения (`a.ordering`). В таком случае происходит переход к третьему условному оператору `if`, где выясняется порядок сортировки: по восходящей или по нисходящей. Если сортировка производится по восходящей, то отображается стрелка, направленная вверх (при вызове метода `orderUpIcon()` из объекта разбиения на страницы) и связываемая с задачей `weblinks.orderup`. Аналогичным образом отображается стрелка, направленная вниз и связываемая с задачей `weblinks.orderdown`.

Ситуация оказывается несколько запутанной, если сортировка производится по нисходящей. Ведь в этом случае порядок расположения элементов в списке изменяется на обратный, а направленная вверх стрелка фактически перемещает элемент вниз по списку. Поэтому во избежание недоразумений направленная вверх стрелка связывается с задачей `weblinks.orderdown`, а направленная вниз стрелка — с задачей `weblinks.orderup`.

На этом выполнение кодового блока внутреннего условного оператора `if` завершается. Далее следуют два условных оператора `endif`, а следовательно, происходит возврат в кодовый блок внешнего условного оператора `if`, где переменной `$disabled` присваивается пустая строка, если переменная `$saveOrder` содержит логическое значение `true`, в противном случае — строка `"disabled='disabled'"`. (Напомним, что строковые значения можно заключать в одинарные кавычки внутри двойных кавычек, и наоборот.) Эта символьная строка используется для отмены доступа к полю ввода порядка сортировки, если сортировка не производится по столбцу упорядочения. Затем отображается само это поле. Но если переменная `$saveOrder` не содержит логическое значение `true`, то поле ввода порядка сортировки окажется недоступным.

Кодовый блок условного оператора `else` выполняется в том случае, если пользователю не разрешено редактировать данный элемент. И тогда порядок сортировки отображается без стрелок или поля его ввода.

Четыре оставшихся столбца таблицы компонуются очень просто. Они служат лишь для отображения уровня доступа, количества посещений, названия конкретного языка (или метки `"All"`, если доступны все поддерживаемые языки), а также идентификатора элемента.

Часть F. Элементы управления разбиением на страницы

Непосредственно перед элементом разметки tbody в рассматриваемом здесь файле исходной компоновки находится следующий код разметки:

```
<tfoot>
    <tr>
        <td colspan="10">
            <?php echo $this->pagination->getListFooter(); ?>
        </td>
    </tr>
</tfoot>
```

В этом коде результат выполнения метода getListFooter() из класса разбиения на страницы помещается в элемент tfoot разметки подзаголовка таблицы. В единственной строке кода создаются все элементы управления, отображаемые в части F рассматриваемой здесь экранной формы, в том числе список Display # (Отобразить страницу по номеру), кнопки Start (В начало), Prev (Назад), Next (Вперед), End (В конец), а также номер страницы! Напомним, что объект разбиения на страницы получается в результате вызова метода getPagination() из модели. Получающийся в итоге экземпляр объекта типа JPagination предоставляет все кнопки и ссылки, необходимые для постраничного перемещения по списку.

Исходное представление и платформа Joomla!

Экранная форма Webinks Manager: Weblinks, приведенная на рис. 8.1, обладает достаточно полными функциональными возможностями и проста в употреблении, позволяя администратору веб-сайта просматривать любое количество элементов веб-ссылок и манипулировать ими. Она содержит немало оперативно доступных средств для фильтрации, сортировки, выполнения групповых операций, переупорядочения и разбиения на страницы. Кроме того, она разработана с учетом защиты от злонамеренного кода, включая регистрацию элементов и снятие их с регистрации, чтобы разные пользователи не пытались одновременно редактировать одни и те же элементы.

Используя встроенные методы, предоставляемые на платформе Joomla!, можно полностью создать аналогичную экранную форму, написав всего лишь около 180 строк относительно простого кода HTML и PHP. Примечательно, что теми же самыми инструментальными средствами можно разрабатывать экранные формы диспетчеров для компонентов, специально создаваемых в виде расширений, как будет показано в главе 10.

Представление одиночных веб-ссылок

Что касается административного компонента com_weblinks, то для полного уяснения особенностей его организации и представления осталось лишь рассмотреть метод display() из класса WeblinksViewWeblink, находящегося в файле views/weblink/view.html.php. Напомним, что данный класс используется при вводе или редактировании отдельных элементов веб-ссылок. Ниже приведен исходный код метода display() из данного класса.

```
public function display($tpl = null)
{
```

```
    $this->state = $this->get('State');
    $this->item = $this->get('Item');
    $this->form = $this->get('Form');

    // проверить ошибки
    if (count($errors = $this->get('Errors'))) {
      JError::raiseError(500, implode("\n", $errors));
      return false;
    }

    $this->addToolbar();
    parent::display($tpl);
}
```

Этот метод аналогичен рассматривавшемуся ранее методу `display()`, за исключением того, что в данном случае из модели вызывается метод `getItem()` вместо метода `getItems()`, а также метод `getForm()` вместо метода `getPagination()`. Напомним, что в методе `JView::get()` из модели вызывается метод, имя которого составляется из префикса `get` и строки аргумента.

Метод `getState()` рассматривался ранее в этой главе, а в методе `getItem()` просто вызывается аналогичный метод `getItem()` из родительского класса `JModelAdmin`. В этом методе выясняется, редактируется ли уже имеющийся элемент. Если он редактируется, то вызывается метод `$table->load($pk)`, где `$pk` — идентификатор элемента веб-ссылки. Затем для столбца `params` таблицы данного элемента создается объект типа `JRegistry`.

Применение класса `JForm` в компоненте Weblinks

В предыдущих главах были представлены примеры применения класса `JForm` для отображения вариантов выбора, например, при редактировании модуля, шаблона или подключаемого модуля. В этих примерах каждый вариант выбора определялся в XML-файле, а сами варианты выбора воспроизводились именно так, как они были определены.

Класс `JForm` можно также использовать для создания сложных форм с гибко настраиваемыми компоновками и динамическими свойствами. На примере компонента Weblinks будет показано, каким образом эти возможности используются для создания формы редактирования отдельных элементов веб-ссылок.

Метод `getForm()` из класса `WeblinksModelWeblink`

Для формы редактирования в методе `getForm()` получается объект типа `JForm`. На рис. 8.2 представлены методы, вызываемые в ходе этого процесса.

Блок-схема выполнения данного метода выглядит немного запутанной, поскольку некоторые методы вызываются из родительских классов. Как было показано на рис. 7.7, класс `WeblinksModelWeblink` расширяет класс `JModelAdmin`, который, в свою очередь, расширяет класс `JModelForm`, а тот — класс `JModel`. В данном примере методы `loadForm()` и `preprocessForm()` вызываются из класса `JModelForm`, тогда как метод `bind()` — из класса `JForm`.

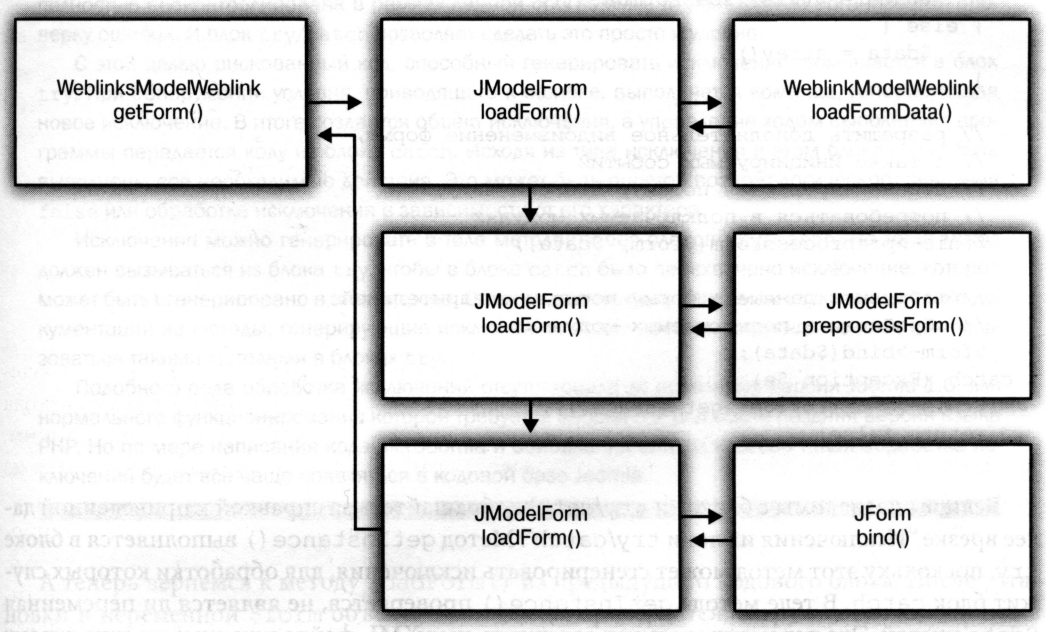


Рис. 8.2. Методы, вызываемые в методе `getForm()` из класса `WeblinksModelWeblink`

Ниже приведена первая часть исходного кода из метода `getForm()`.

```
public function getForm($data = array(), $loadData = true)
{
  // инициализировать переменные
  $app = JFactory::getApplication();

  // получить форму
  $form = $this->loadForm('com_weblinks.weblink', 'weblink',
➥array('control' => 'jform', 'load_data' => $loadData));
  if (empty($form)) {
    return false;
}
```

В строке кода, выделенной выше полужирным, вызывается метод `loadForm()` из родительского класса `JModelForm`. В качестве первого аргумента при вызове этого метода указывается переменная `$name` со значением `"com_weblinks.weblink"`, в качестве второго аргумента — переменная `$source` со значением `"weblink"`, в качестве третьего аргумента — ассоциативный массив, инициируемый непосредственно в данной строке кода. В методе `loadForm()` проверяется, была ли форма уже кеширована. Если она не кеширована, то выполняется приведенный ниже код из блока `try/catch`.

```
try {
    $form = JForm::getInstance($name, $source, $options, false, $xpath);

    if (isset($options['load_data']) && $options['load_data']) {
        // получить данные для формы
```

```
            $data = $this->loadFormData();
        } else {
            $data = array();
        }

        // разрешить дополнительное видоизменение формы,
        // а также инициируемых событий
        // Данные передаются потому, что они могут
        // потребоваться в подключаемых модулях.
        $this->preprocessForm($form, $data);

        // загрузить данные в форму после предварительной
        // обработки в подключаемых модулях
        $form->bind($data);
    } catch (Exception $e) {
        $this->setError($e->getMessage());
        return false;
    }
```

Если вы незнакомы с блоками `try/catch`, обращайтесь за справкой к приведенной далее врезке "Исключения и блоки `try/catch`". Метод `getInstance()` выполняется в блоке `try`, поскольку этот метод может сгенерировать исключения, для обработки которых служит блок `catch`. В теле метода `getInstance()` проверяется, не является ли переменная `$data` пустой. Эта переменная может содержать имя XML-файла или символьную строку с кодом XML для формы. Далее в методе `getInstance()` проверяется первый символ из переменной `$data`. Если это символ <, то предполагается, что в данной переменной содержится символьная строка с кодом XML, а далее предпринимается попытка создать форму с помощью метода `load()` из класса `JForm`. В данном примере переменная `$data` содержит имя XML-файла `weblink`, и поэтому вместо метода `load()` из класса `JForm` для загрузки формы используется этот файл, а именно `models/forms/weblink.xml`. В обоих методах форма создается из элементов `fieldset` и `field` разметки в коде XML. В файле `weblink.xml` определяются два элемента `fieldset`, содержащие ряд элементов разметки `field`. Более подробно применение элементов разметки полей для воспроизведения формы будет показано далее в этой главе, когда будут рассматриваться файлы компоновки экранной формы редактирования.

Все возникающие ошибки полностью обрабатываются в методе `getInstance()`. Нужно лишь сгенерировать исключение соответствующего типа для каждого условия возникновения ошибки. А исключения обрабатываются в блоке `catch` вызывающего метода, как показано в приведенном выше коде. В данном случае характер ошибки устанавливается, исходя из сообщения, получаемого от объекта исключения, а затем возвращается логическое значение `false`.

Исключения и блоки `try/catch`

В версии 5 языка PHP введено понятие обработки исключений, а также внедрены средства их генерирования и перехвата в блоках `try/catch`. Аналогичные понятия и средства имеются в Java и других языках ООП. Основной принцип обработки исключений заключается в следующем: если в программе выполняется какая-нибудь рискованная операция, которая не может быть

полностью проконтролирована в рамках данной программы, то необходимо организовать проверку ошибок. И блок `try/catch` позволяет сделать это просто и удобно.

С этой целью рискованный код, способный генерировать исключения, помещается в блок `try`. При обнаружении условия, приводящего к ошибке, выполняется команда, генерирующая новое исключение. В итоге создается объект исключения, а управление ходом выполнения программы передается коду из блока `catch`. Исходя из типа исключения в этом блоке могут быть выполнены все необходимые действия. Это может быть простой возврат логического значения `false` или обработка исключения в зависимости от его характера.

Исключения можно генерировать в теле методов и без блоков `try/catch`. Но тогда метод должен вызываться из блока `try`, чтобы в блоке `catch` было перехвачено исключение, которое может быть сгенерировано в этом методе. Об этом должно быть непременно указано в блоке документации на методы, генерирующие исключения, чтобы программирующие знали, как пользоваться такими методами в блоках `try`.

Подобного рода обработка исключений отсутствовала до появления версии Joomla 1.6, для нормального функционирования которой требуется версия 5.2 или более поздняя версия языка PHP. Но по мере написания кода для Joomla и обновления старых классов такая обработка исключений будет все чаще появляться в кодовой базе Joomla.

А теперь вернемся к методу `loadForm()` из предыдущего кодового блока. После установки в переменной `$form` объекта типа `JForm` проверяется, загружаются ли данные в форму. В качестве третьего аргумента в сигнатуре метода `loadForm()` указывается ассоциативный массив `$options`. В данном примере требуется предварительно загрузить данные, предназначенные для формы, для чего в ассоциативный массив вводится элемент `load_data` с логическим значением `true`.

При соблюдении условия загрузки данных вызывается метод `loadFormData()`, получающий любые загруженные ранее данные, но, как правило, они отсутствуют. Если элемент редактируется в первый раз или только создается, то вызывается метод `$this->getItem()`, в котором создается объект типа `JObject` с полями для формы. И если элемент уже существует, то для него получаются данные, тогда как для нового элемента получаются значения, исходные по умолчанию. А если таковые не указаны в полях, то получаются пустые значения.

Когда же используются данные из текущего сеанса работы? Допустим, пользователь заполнил форму с целью ввести новый элемент, но некоторые данные в этой форме оказались недостоверными, например, пропущено заглавие или продублировано поле псевдонима. В этом случае операция сохранения не завершится успешно, а следовательно, данные не сохранятся в базе данных. И тогда пользователю предстоит неприятная процедура повторного ввода данных во всех полях формы, чтобы исправить единственную допущенную им ошибку. Во избежание подобной ситуации введенные данные сохраняются в текущем сеансе работы пользователя. И только после успешного их сохранения в базе данных они удаляются из данного сеанса работы. Таким образом, данные получаются из базы данных с помощью метода `getItem()` или же из текущего сеанса работы с помощью метода `getUserState()`.

И последней операцией в методе `loadFormData()` является проверка наличия категории в запросе или сеансе работы. Если категория имеется, она устанавливается для данных как исходная по умолчанию. И наконец, из данного метода возвращается объект типа

JObject, сохраняемый в переменной $data из вызывающего метода loadFormData(). Если же загружать данные в форму не требуется, в переменной $data устанавливается пустой массив.

Следующая строка кода из метода loadFormData() имеет большое значение:

```
$this->preprocessForm($form, $data);
```

В этой строке кода вызывается метод из класса JModelForm, поскольку он не переопределяется в подклассах, производных от класса JModelAdmin или WeblinksModelWeblink. Ниже приведен исходный код данного метода.

```
protected function preprocessForm(JForm $form, $data, $group = 'content')
{
  // импортировать подходящую группу подключаемых модулей
  JPluginHelper::importPlugin($group);

  // получить диспетчер
  $dispatcher = JDispatcher::getInstance();

  // инициировать событие, связанное с подготовкой формы
  $results = $dispatcher->trigger('onContentPrepareForm', array($form,
↳$data));

  // проверить ошибки, возникающие при подготовке формы
  if (count($results) && in_array(false, $results, true)) {
    // получить последнюю ошибку
    $error = $dispatcher->getError();

    // преобразовать ошибку, если требуется, в исключение
    if (!JError::isError($error)) {
      throw new Exception($error);
    }
  }
}
```

В данном методе инициируется событие onContentPrepareForm. Это событие было внедрено в версии Joomla 1.6 специально для объектов типа JForm. Оно позволяет разрабатывать подключаемые модули для видоизменения объектов типа JForm, прежде чем они будут воспроизведены в окне браузера. Такие подключаемые модули делают доступным объект типа JForm (в переменной $form) и массив данных (в переменной $data). Пример применения события onContentPrepareForm был приведен в главе 5 при создании подключаемого модуля из файла myregistration2.php. В том примере подключаемый модуль использовался для ввода в форму ряда новых полей. Но с тем же успехом подключаемые модули можно использовать и для удаления, проверки достоверности или редактирования объектов типа JForm.

После выполнения подключаемых модулей проверяются возможные ошибки. Если подключаемый модуль возвратил ошибку, не являющуюся объектом типа Exception, она преобразуется в этот объект, который далее используется в команде генерирования исключения с целью сообщить об ошибке вызывающему методу.

А после возврата обратно в метод loadForm() вызывается метод bind() из родительского класса JForm, расположенного вверх по иерархии, поскольку этот метод не

переопределяется ни в одном из его подклассов. В данном методе проверяется наличие в объекте типа JForm достоверного объекта типа JXMLElement, определенного в поле, размеченном в коде XML, а также наличие объекта или массива в переменной $data. После этого поля формы сопоставляются с элементами данных и в каждом из них устанавливается соответствующее значение.

И снова происходит возврат в метод loadForm() ближе к концу блока try. Если при выполнении кода в любом из методов было сгенерировано исключение, его необходимо обработать. С этой целью выполняется приведенный ниже блок catch.

```
} catch (Exception $e) {
        $this->setError($e->getMessage());
        return false;
    }
```

В этом кодовом блоке характер ошибки устанавливается для данной модели, исходя из сообщения, получаемого от объекта исключения, а затем возвращается логическое значение false вызывающему методу loadForm(). Если же во время выполнения блока try исключения не сгенерированы, блок catch пропускается.

Сохранение объекта типа JForm в оперативной памяти

В отношении метода loadForm() остается лишь заметить, что перед выполнением блока try необходимо убедиться, сохранена ли форма в оперативной памяти. Такая проверка производится в следующем фрагменте кода:

```
// сформировать хеш-значение для электронной подписи
$hash = md5($source.serialize($options));

// проверить, можно ли воспользоваться загруженной ранее формой
if (isset($this->_forms[$hash]) && !$clear) {
    return $this->_forms[$hash];
}
```

У этого фрагмента кода очень простое назначение: форму необходимо сохранить на тот случай, если она понадобится снова в данном цикле запроса. Ведь извлечь форму из оперативной памяти намного быстрее, чем воссоздавать ее из XML-файла. А поскольку в одном цикле запроса могут оказаться активными несколько объектов типа JForm, то для данной конкретной формы необходимо сформировать однозначный идентификатор. С этой целью вызываются функции md5() и serialize() языка PHP. Так, функция serialize() преобразует объект в символьную строку, сохраняя всю информацию, необходимую для обратного преобразования в исходный объект. А функция md5() принимает любую символьную строку в качестве аргумента и формирует 32-символьную строку, однозначно определяющую исходную строку. Символьная строка, возвращаемая функцией md5(), служит в качестве ключа к ассоциативному массиву для сохранения в нем сериализованного (последовательно упорядоченного) объекта. Такой алгоритм дает следующие преимущества.

- Один и тот же экземпляр объекта типа JForm будет всегда давать то же самое хеш-значение.

- Значение будет всегда однозначно определять данный объект типа JForm, а следовательно, в оперативной памяти можно хранить несколько объектов данного типа, используя для этого отдельный массив.
- Хеш-значение пригодно для хранения в ассоциативном массиве.

В данном примере переменная $options содержит ассоциативный массив с двумя элементами: control со значением "jform" и load_data со значением **1**, а переменная $source — значение "weblink". Вызов функции serialize($options) приводит к следующему результату:

```
a:2:{s:7:"control";s:5:"jform";s:9:"load_data";b:1;}
```

а вызов функции $source.serialize($options) — к такому:

```
weblinka:2:{s:7:"control";s:5:"jform";s:9:"load_data";b:1;}
```

И наконец, в результате вызова функции md5($source.serialize($options)) переменной $hash присваивается следующее хеш-значение:

```
37ceb359374f97c561de59903862618e
```

Получив хеш-значение в переменной $hash, можно далее проверить, сохранена ли данная конкретная форма в поле $this->_forms. Если она сохранена, то это означает, что объект именно этой формы уже находится в оперативной памяти, а следовательно, данную форму уже не придется загружать из XML-файла. И как только объект данной формы будет получен, остальная часть метода loadForm() пропускается и происходит возврат из него.

Когда достоверная форма уже имеется, после блока try/catch следует приведенная ниже строка кода с комментарием.

```
// сохранить форму на будущее
$this->_forms[$hash] = $form;
```

В этой строке кода объект нужной формы сохраняется в массиве, оставаясь там до тех пор, пока он не понадобится снова. Рассмотренный здесь способ сохранения объектов в ассоциативных массивах с помощью хеш-значения в качестве ключа к массиву нередко применяется для экономии времени, затрачиваемого на обработку данных в различных местах Joomla.

Видоизменение форм в динамическом режиме

Итак, объект типа JForm загружен в оперативную память с помощью метода getForm() из класса WeblinksModelWeblink. А теперь рассмотрим пример того, как можно видоизменить форму в самом коде перед ее отображением. В остальной части исходного кода метода getForm() из класса WeblinksModelWeblink можно обнаружить целый ряд вызовов метода $form->setFieldAttribute(), в том числе и в приведенных ниже строках кода.

```
// видоизменить форму, опираясь на средства контроля доступа
if (!$this->canEditState((object) $data)) {
    // сделать поля недоступными для отображения
    $form->setFieldAttribute('ordering', 'disabled', 'true');
```

```
        $form->setFieldAttribute('state', 'disabled', 'true');
        $form->setFieldAttribute('publish_up', 'disabled', 'true');
        $form->setFieldAttribute('publish_down', 'disabled', 'true');

        // сделать поля недоступными при сохранении
        // В контроллере уже проверено, что эту запись можно править.
        $form->setFieldAttribute('ordering', 'filter', 'unset');
        $form->setFieldAttribute('state', 'filter', 'unset');
        $form->setFieldAttribute('publish_up', 'filter', 'unset');
        $form->setFieldAttribute('publish_down', 'filter', 'unset');
}
```

Метод `setFieldAttribute()` относится к числу самых эффективных методов, предназначенных для видоизменения формы в динамическом режиме.

В приведенном выше фрагменте кода проверяется, разрешено ли пользователю редактировать элемент веб-ссылки, но не изменять его состояние. Изменение состояния элемента означает смену его состояния "опубликовано", а также видоизменение другой информации, связанной с публикацией данного элемента. В данном случае поля формы должны по-прежнему отображаться, но быть недоступными для пользователя. Так, если пользователю запрещено изменять состояние данного элемента, поля формы следует сделать недоступными, установив логическое значение `true` в атрибуте `disabled` разметки этих полей. И при отображении формы эти поля окажутся недоступными для редактирования. Таким образом, значения этих атрибутов, установленные в XML-файле разметки данной формы, переопределяются динамически.

Вторая часть рассматриваемого здесь фрагмента кода имеет отношение к сохранению информации в базе данных. Как следует из приведенных выше примеров, организуя защиту данных, нельзя полагаться на HTML-формы. Ведь злоумышленник может легко отредактировать форму, сделав, например, недоступное поле доступным для редактирования. Поэтому в классе `JForm` имеется метод фильтрации, вызываемый при установке атрибута `filter`. В данном случае в этом атрибуте устанавливается значение `"unset"`, что, в свою очередь, приводит к установке пустого значения в данном поле формы. Таким образом, в данном поле используется текущее значение (для существующего элемента веб-ссылки) или устанавливаемое по умолчанию значение.

Подобным способом можно очень просто настраивать формы динамически перед их отображением, а затем и в процессе сохранения. Как упоминалось ранее, такой же способ можно применять и в специальном расширении подключаемого модуля для динамического видоизменения основных форм, используя для этой цели событие `onContentPrepareForm`.

Воспроизведение формы

На последней стадии рассматриваемого здесь процесса форма воспроизводится в окне браузера. Для этой цели в папке `views/weblink/tmpl` с файлами компоновки компонента Weblinks имеются файлы `edit.php` и `edit_params.php`. В этих файлах форма компонуется с помощью объекта типа `JForm`. В частности, файл `edit.php` служит для компоновки основной части формы, тогда как файл `edit_params.php`, как и следовало ожидать, — для отображения параметров элемента веб-ссылки.

Файл edit.php

Пример этого файла показывает, насколько просто отображаются объекты типа JForm. Ниже приведена первая часть файла edit.php.

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

JHtml::addIncludePath(JPATH_COMPONENT.'/helpers/html');
JHtml::_('behavior.tooltip');
JHtml::_('behavior.formvalidation');
?>
<script type="text/javascript">
    Joomla.submitbutton = function(task)
    {
      if (task == 'weblink.cancel' ||
↳document.formvalidator.isValid(document.id('weblink-form'))) {
        <?php echo $this->form->getField('description')->save(); ?>
        Joomla.submitform(task, document.getElementById('weblink-form'));
      }
      else {
        alert('<?php echo $this->escape
↳(JText::_('JGLOBAL_VALIDATION_FORM_FAILED'));?>');
      }
    }
</script>
```

Как и в компоновку, рассматривавшуюся ранее в этой главе, в данную компоновку включена папка helpers для доступа к специальным методам из класса JHtml. Затем в данную компоновку добавляются два вида поведения: всплывающая подсказка и проверка достоверности формы. В частности, класс JHtmlBehavior содержит методы для вызова функций JavaScript, а метод formvalidation() позволяет запустить сценарий JavaScript для проверки достоверности формы, исходя из атрибутов, установленных в объекте типа JForm.

В следующем кодовом блоке содержится элемент разметки script, в котором вводится сценарий JavaScript, запускаемый для проверки достоверности формы. Более подробно этот код будет обсуждаться в главе 12. Далее следует приведенный ниже код компоновки, в котором начинается воспроизведение данной формы.

```
<form action="<?php echo
↳JRoute::_('index.php?option=com_weblinks&layout=edit&id='.(int)
↳$this->item->id); ?>" method="post" name="adminForm" id="weblink-form"
↳class="form- validate">
  <div class="width- 60 fltlft">
    <fieldset class="adminform">
      <legend><?php echo empty($this->item->id) ?
↳JText::_('COM_WEBLINKS_NEW_WEBLINK') : JText::sprintf(
↳'COM_WEBLINKS_EDIT_WEBLINK', $this->item->id); ?></legend>
    <ul class="adminformlist">
    <li><?php echo $this->form->getLabel('title'); ?>
    <?php echo $this->form->getInput('title'); ?></li>

    <li><?php echo $this->form->getLabel('alias'); ?>
    <?php echo $this->form->getInput('alias'); ?></li>
```

```
    <li><?php echo $this->form->getLabel('catid'); ?>
    <?php echo $this->form->getInput('catid'); ?></li>

    <li><?php echo $this->form->getLabel('state'); ?>
    <?php echo $this->form->getInput('state'); ?></li>

    <li><?php echo $this->form->getLabel('access'); ?>
    <?php echo $this->form->getInput('access'); ?></li>

    <li><?php echo $this->form->getLabel('ordering'); ?>
    <?php echo $this->form->getInput('ordering'); ?></li>

    <li><?php echo $this->form->getLabel('language'); ?>
    <?php echo $this->form->getInput('language'); ?></li>

    <li><?php echo $this->form->getLabel('id'); ?>
    <?php echo $this->form->getInput('id'); ?></li>
    </ul>

    <?php echo $this->form->getLabel('description'); ?>
    <div class="clr"></div>
    <?php echo $this->form->getInput('description'); ?>
    </fieldset>
</div>
```

В первой строке приведенного выше фрагмента кода создается элемент `form` разметки формы в коде HTML с атрибутом `action`, направляющим по тому же самому URL обратно в исходное место. Здесь же определяется метод отправки для сохранения полей формы в переменной `$_REQUEST`. Затем создается элемент разметки `div`, а в нем отображается элемент разметки `legend`. Если имеется элемент с идентификатором `id`, то редактируется уже существующая веб-ссылка, в противном случае вводится новая веб-ссылка.

Далее открывается элемент `ul` разметки неупорядоченного списка. В этом элементе начинается воспроизведение полей формы, каждое из которых отображается в элементе `li` разметки отдельной позиции в списке. Для каждого поля выводится метка по команде `echo`, вызывающей метод `getInput()`, причем обращение к каждому полю осуществляется по его имени. С помощью всего двух строк кода, приходящихся на каждое поле, можно воспроизвести всю форму в любом порядке ее компоновки.

Следует заметить, что для данной компоновки совсем не обязательно знать тип воспроизводимого поля. На рис. 8.3 приведена экранная форма редактирования отдельных веб-ссылок. Заглавие, псевдоним и URL представлены в этой форме полями ввода **Title**, **Alias** и **URL** соответственно и помечены знаком звездочки (*) как данные, обязательные для ввода. А поля категории, состояния, упорядочения и языка представлены раскрывающимися списками **Category**, **Status**, **Ordering** и **Language** соответственно. В нижней части данной формы, обозначенной меткой **Description**, представлены кнопки редактора, а также область, в которой пользователь может ввести подробное описание веб-ссылки. Каждое из этих полей объекта типа `JForm` воспроизводится с помощью тех же самых двух строк кода, в которых вызываются методы `getLabel()` и `getInput()`.

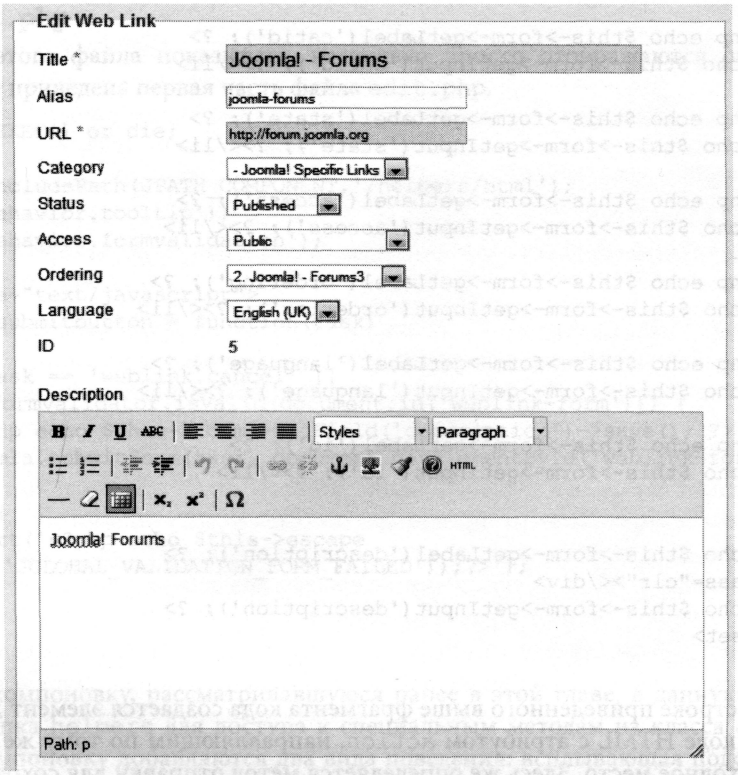


Рис. 8.3. Экранная форма редактирования отдельных веб-ссылок

В следующей части кода компоновки отображаются так называемые слайдеры, располагаемые в правой части данной формы. Они представляют собой разворачиваемые и сворачиваемые области для показа или сокрытия различных групп полей. В качестве примера ниже приведен код для компоновки области Publishing Options (Параметры публикации) данной формы.

```
<div class="width-40 fltrt">
  <?php echo JHtml::_('sliders.start','newsfeed-sliders-'.
↳$this->item->id, array('useCookie'=>1)); ?>

  <?php echo JHtml::_('sliders.panel',
↳JText::_('JGLOBAL_FIELDSET_PUBLISHING'), 'publishing-details'); ?>

  <fieldset class="panelform">
    <ul class="adminformlist">
      <li><?php echo $this->form->getLabel('created_by'); ?>
      <?php echo $this->form->getInput('created_by'); ?></li>

      <li><?php echo $this->form->getLabel('created_by_alias'); ?>
      <?php echo $this->form->getInput('created_by_alias'); ?></li>

      <li><?php echo $this->form->getLabel('created'); ?>
```

```
      <?php echo $this->form->getInput('created'); ?></li>

      <li><?php echo $this->form->getLabel('publish_up'); ?>
      <?php echo $this->form->getInput('publish_up'); ?></li>

      <li><?php echo $this->form->getLabel('publish_down'); ?>
      <?php echo $this->form->getInput('publish_down'); ?></li>

      <?php if ($this->item->modified_by) : ?>
         <li><?php echo $this->form->getLabel('modified_by'); ?>
         <?php echo $this->form->getInput('modified_by'); ?></li>

         <li><?php echo $this->form->getLabel('modified'); ?>
         <?php echo $this->form->getInput('modified'); ?></li>
      <?php endif; ?>

      <?php if ($this->item->hits) : ?>
         <li><?php echo $this->form->getLabel('hits'); ?>
         <?php echo $this->form->getInput('hits'); ?></li>
       <?php endif; ?>

    </ul>
</fieldset>
```

В первой строке приведенного выше фрагмента кода вызывается метод `JHtmlSliders::start()` для формирования области слайдеров. А в следующей строке вызывается метод `JHtmlSliders::panel()` для создания панели со слайдерами. Все поля, размечаемые в элементе `fieldset`, отображаются в пределах данной панели. Функции слайдеров реализуются средствами JavaScript и MooTools, подробнее рассматриваемыми в главе 12. Но самое замечательное, что все эти функции можно реализовать, вызывая упомянутые выше методы.

Далее создается элемент `fieldset` для разметки полей, как и прежде. А сами поля воспроизводятся с помощью методов `getLabel()` и `getInput()`. Однако для воспроизведения трех последних полей код их разметки помещается в блоке условного оператора `if`. Но если в этих полях нечего показывать, то они и не отображаются.

После этого в файле компоновки `edit.php` следует приведенная ниже строка кода.

```
<?php echo $this->loadTemplate('params'); ?>
```

В этой строке кода осуществляется поиск файла компоновки `views/weblink/tmpl/edit_params.php` и затем отображается загружаемая из него компоновка. Открыв этот файл, можно обнаружить следующий фрагмент кода компоновки:

```
  $fieldSets = $this->form->getFieldsets('params');
  foreach ($fieldSets as $name => $fieldSet) :
    echo JHtml::_('sliders.panel',JText::_($fieldSet->label),
↳$name.'-params');
      if (isset($fieldSet->description) && trim($fieldSet->description)) :
        echo '<p class="tip">'.
↳$this->escape(JText::_($fieldSet->description)).'</p>';
      endif;
      ?>
```

```
    <fieldset class="panelform">
      <ul class="adminformlist">
      <?php foreach ($this->form->getFieldset($name) as $field) : ?>
        <li><?php echo $field->label; ?>
        <?php echo $field->input; ?></li>
      <?php endforeach; ?>
      </ul>
    </fieldset>
<?php endforeach; ?>
```

В этом фрагменте кода демонстрируется краткая форма воспроизведения полей из объекта типа `JForm`. В данном случае все поля воспроизводятся по порядку их следования в неупорядоченном списке. Вместо написания двух строк кода для воспроизведения каждого поля здесь используется циклическое обращение ко всем полям из объекта типа `JForm`. Для этой цели организуются два цикла. Во внешнем цикле происходит поочередное обращение ко всем элементам разметки `fieldset` в массиве, получаемом с помощью метода `getFieldSets('params')` из всех наборов полей объекта типа `JForm`, сгруппированных под общим названием `params`. Напомним, что в XML-файле разметки формы, представленной объектом типа `JForm`, элементы `fieldset` можно дополнительно размещать в элементах `fields`. Так, в файле `weblink.xml` параметры размещаются в элементе разметки `fields` под названием `params`. Сначала для каждого набора полей создается панель ползунков с соответствующей меткой и описанием, отображаемым во всплывающей подсказке, а затем открывается элемент разметки `fieldset` и вложенные в него элементы разметки `ul`.

Далее начинается внутренний цикл, где обрабатывается каждое поле. Для получения массива полей данного набора вызывается метод `getFieldSet($name)`, где `$name` — атрибут `name` элемента разметки `fieldset`. Затем создается элемент разметки `li` и снова по команде `echo` выводится метка поля и введенное в нем значение. Но в данном случае объект типа `JFormField` уже содержится в переменной `$field`, и поэтому вместо вызова методов `getLabel()` и `getField()` из класса `JForm` используется непосредственное обращение к метке поля `$field->label` и введенному в нем значению `$field->input`.

Преимущество организации вложенных друг в друга циклов `foreach` заключается в том, что в такой код не придется вносить изменения, если потребуется добавить или удалить элементы разметки наборов полей и отдельных полей в XML-файле. Содержимое любых элементов разметки `fieldset` и `field` в элементе `fields` под названием `params` будет воспроизведено в точном соответствии с их расположением в XML-файле. Таким способом воспроизводятся поля для ввода или выбора параметров модулей, шаблонов, подключаемых модулей и языковых расширений. Аналогичным способом вводятся или выбираются параметры компонентов, только в данном случае каждый элемент разметки `fieldset` создает вкладку в режимном окне экранной формы для ввода параметров.

В состав класса `JForm` входит немало других методов, предназначенных для манипулирования формой. Это дает разработчикам возможность подстраивать форму под отдельных пользователей и разные ситуации.

Краткие итоги рассмотрения административной части компонента Weblinks

Итак, мы рассмотрели все самые важные составляющие административной части компонента com_weblinks, в том числе установочные файлы, вспомогательные методы и файлы параметров. Этот компонент был выбран в качестве примера, чтобы стали понятнее особенности построения компонентов на основе шаблона проектирования MVC, а также обработки задач в контроллере и применения в нем моделей и представлений, необходимых для обеспечения нормальной работы компонента.

Пользовательская часть компонента Weblinks

А теперь рассмотрим пользовательскую часть компонента Weblinks. Уделим основное внимание сходствам и отличиям пользовательской части этого компонента от административной, поскольку последняя уже была рассмотрена достаточно подробно.

Сходство структуры файлов и применения шаблона MVC

В табл. 8.1 перечислены папки и файлы из верхнего уровня структуры файлов пользовательского компонента Weblinks. Эти файлы находятся в папке components/com_weblinks. (Здесь и далее они указываются относительно именно этой папки.)

Таблица 8.1. Папки и основные файлы пользовательской части компонента Weblinks

Имя папки или файла	Содержимое	Категория по шаблону MVC
controllers	Контроллер для экранной формы ввода данных: файл weblink.php	Контроллер
helpers	Вспомогательные файлы: category.php, icon.php, route.php	Вспомогательные средства/Разное
models	Модели: файлы categories.php, category.php, form.php, weblink.php	Модель
modesl/forms	XML-файл для разметки формы, представленной объектом типа JForm: weblink.xml	Модель
views/categories/tmpl	Представление пункта меню Categories: файл view.html.php	Представление
views/categories/tmp	Файлы компоновки пункта меню Categories; XML-файл для разметки вариантов выбора данного пункта меню	Представление
views/category	Представления пункта меню отдельной категории: файлы view.html.php и view.feed.php	Представление
views/category/tmpl	Файлы компоновки пункта меню отдельной категории; XML-файл для разметки вариантов выбора данного пункта меню	Представление
views/form	Представление пункта меню для передачи веб-ссылки: файл view.html.php	Представление

Окончание табл. 8.1

Имя папки или файла	Содержимое	Категория по шаблону MVC
`views/form/tmpl`	Файлы компоновки формы для ввода данных; XML-файл для разметки вариантов выбора данного пункта меню	Представление
`controller.php`	Контроллер, используемый по умолчанию для выполнения задачи отображения	Контроллер
`router.php`	Маршрутизатор компонента	Вспомогательные средства/Разное
`weblinks.php`	Сценарий, служащий точкой входа в компонент	Контроллер

Как видите, организационная структура папок в пользовательской части очень похожа на их организацию в административной части компонента Weblinks. Как и в административной части, здесь имеются отдельные папки для контроллеров, вспомогательных средств, моделей и представлений, но в то же время отсутствует установочная папка. В пользовательской части данного компонента требуется лишь один ряд установочных файлов, которые принято хранить в административной части. Здесь также отсутствует папка `tables`. Ведь класс для таблицы `#__weblinks` достаточно определить лишь один раз, и это делается опять же в административной части.

Открыв файл `weblinks.php` на верхнем уровне структуры папок пользовательской части компонента Weblinks, можно обнаружить приведенный ниже исходный код, предназначенный для управления этим компонентом.

```
$controller = JController::getInstance('Weblinks');
$controller->execute(JRequest::getCmd('task'));
$controller->redirect();
```

Это, по существу, те же самые строки кода, что и в административной части данного компонента, а следовательно, реализуемый в них процесс действует таким же образом. Сначала получается контроллер, подходящий для выполнения указанной задачи, а затем выполняется сама задача. Если задача отличается от обычного отображения, то устанавливается и происходит переадресация к нужной экранной форме по соответствующему URL. А если это задача отображения, то просто отображается страница, а переадресация вообще не устанавливается.

Как и в административной части данного компонента, в пользовательской его части у каждого представления имеется своя модель. В данном случае имеются следующие три представления.

- `Categories`. Отображает все категории веб-ссылок в иерархическом списке.
- `Category`. Отображает веб-ссылки в одной категории.
- `Form`. Отображает форму предъявления веб-ссылки.

У каждого из этих представлений имеется своя модель, класс представления, расширяющий класс `JView`, а также один или более файл компоновки, включаемый в класс представления с помощью метода `loadTemplate()`, вызываемого из класса `JView`.

Два представления (Categories и Category) служат лишь для отображения информации, поэтому ими оперирует базовый контроллер из класса WeblinksController, находящегося в классе controller.php. А представление Form дает пользователю возможность вводить и редактировать отдельные веб-ссылки. Задачи, не связанные с отображением формы в этом представлении, обрабатываются контроллером из класса WeblinksControllerWeblink. Для этой формы предназначаются задачи ввода, отмены, редактирования и сохранения. Как и в административной части данного компонента, задачи ввода, отмены и редактирования обрабатываются контроллером, а для сохранения информации в базе данных в одном методе save() вызывается другой метод save() из соответствующей модели.

Структура файлов рассматриваемого здесь компонента отличается следующей примечательной особенностью: в папке models/weblink находится модель класса WeblinksModelWeblink. Но ведь класс модели формы WeblinksModelForm расширяет класс WeblinksModelWeblink из *административной* части, а не класс WeblinksModelWeblink из пользовательской части данного компонента. В начале определения класса WeblinksModelForm находится следующая строка кода:

```
require_once JPATH_COMPONENT_ADMINISTRATOR.'/models/weblink.php';
```

А далее следует приведенная ниже строка кода.

```
class WeblinksModelForm extends WeblinksModelWeblink
```

Для расширения класса WeblinksModelWeblink используется код из файла, относящегося к административной части, поскольку этот класс загружается из папки administrator/components/models/weblink.php. А класс WeblinksModelForm наследует метод save() из класса JModelAdmin.

Класс WeblinksModelWeblink из пользовательской части задействуется при выборе элемента веб-ссылки щелчком кнопкой мыши. Элемент веб-ссылки отличается от других элементов содержимого. Он отображает внешнюю веб-ссылку, а не ее компоновку в пределах веб-сайта. Следовательно, представление и компоновка для отдельного элемента веб-ссылки отсутствуют. Зачем же тогда нужен компонент Weblinks, если внешние URL можно отображать вообще без компонента? Дело в том, что компонент позволяет делать то, чего нельзя сделать с помощью одной лишь внешней ссылки, в том числе следующее:

- организовать ссылки по отдельным категориям;
- расположить ссылки по категориям;
- ограничить уровни доступа к ссылкам;
- подсчитывать количество посещений по ссылкам;
- разрешить пользователям передавать ссылки;
- публиковать ссылки и снимать их с публикации.

Ниже приведен пример URL, сформированный для элемента веб-ссылки.

```
<URL веб-сайта>/index.php/using-joomla/extensions/components/weblinkscomponent/
weblinks-single-category?task=weblink.go&id=5
```

Обратите внимание на то, что в этом URL указаны идентификатор элемента веб-ссылок и конкретная задача weblink.go. Как пояснялось ранее, при переходе по этому

URL выполняется метод `go()` из класса `WeblinksControllerWeblink`. Именно здесь и проверяется состояние "опубликовано", уровень доступа для пользователя и приращение счетчика посещений.

Для оказания помощи при выполнении подобных задач используется класс `WeblinksModelWeblink` из пользовательской части данного компонента. В частности, метод `getItem()` вызывается из модели для извлечения элемента веб-ссылки из базы данных, а метод `hit()` — для приращение счетчика посещений этого элемента, определяющего, сколько раз осуществлялся доступ к данному элементу. Если все эти задачи выполняются успешно, то устанавливается переадресация по внешнему URL, который отображается в цикле переадресации. Обратите внимание на то, что эта последовательность операций отличается от той, что выполнялась в административной части данного компонента. Там переадресация устанавливалась для обратной ссылки на экранную форму в компоненте Weblinks (например, на список или экранную форму редактирования веб-ссылок в компоненте Weblinks Manager).

Типы пунктов меню

К числу важных отличий пользовательской части веб-сайта на Joomla от административной относится порядок создания пунктов (или вариантов выбора из) меню. Напомним, что в административной части три пункта меню Components были созданы на основании информации, извлекаемой из элементов разметки `menu` и `submenu` в установочном XML-файле `administrator/components/com_weblinks/weblinks.xml`. Все задачи, выполняемые администратором веб-сайта с помощью компонента Weblinks, заранее известны, и поэтому для их поддержки можно сформировать соответствующие пункты меню.

В пользовательской части веб-сайта меню и их пункты полностью определяются самим пользователем. Так, на одном веб-сайте могут вообще отсутствовать меню и их пункты для выбора компонента Weblinks, тогда как на другом — присутствовать многие пункты меню для выбора различных категорий компонента Weblinks. Для поддержки подобных возможностей данный компонент предоставляет администратору веб-сайта специальные инструментальные средства, с помощью которых он может создавать пункты меню и манипулировать ими по своему усмотрению.

Список доступных типов пунктов меню составляется в результате анализа всех XML-файлов, находящихся в папках компоновки каждого компонента. Так, для компонента Weblinks имеются три XML-файла, предназначенные для создания трех типов пунктов меню, как показано в табл. 8.2.

Таблица 8.2. Типы пунктов меню для выбора веб-ссылок

Папка	Имя файла	Тип пункта меню
`views/categories/tmpl`	`default.xml`	Перечисление всех категорий веб-ссылок
`views/category/tmpl`	`default.xml`	Перечисление всех веб-ссылок данной категории
`views/form/tmpl`	`edit.xml`	Предъявление веб-ссылки

Как пояснялось в главе 4, альтернативные пункты меню создаются копированием этих базовых XML-файлов и файлов компоновки в папку `html` шаблона и последующим их

видоизменением по мере надобности. Структура основных XML-файлов такая же, как и у XML-файлов замещения и других видов расширений. Внешним элементом разметки в них является элемент `metadata`, в который включается элемент разметки `layout` с языковыми ключами для заглавия и варианта выбора. Эти ключи определяются в файле `administrator/language/en-GB/en-GB.com_weblinks.sys.ini` и отображаются вместе со списком типов пунктов меню. Как пояснялось ранее, имеется также возможность определить элемент разметки `help`. А в элементе разметки `message` отображается описание компоновки (оно появляется во всплывающей подсказке при просмотре списка типов пунктов меню).

Ниже в качестве примера приведен элемент `layout` для разметки типа пункта меню, предназначенного для выбора единственной категории веб-ссылок.

```
<layout title="com_weblinks_category_view_default_title"
↳option="com_weblinks_category_view_default_option">
  <help
    key="JHELP_MENUS_MENU_ITEM_WEBLINK_CATEGORY"
  />
  <message>
    <![CDATA[com_weblinks_category_view_default_desc]]>
  </message>
</layout>
```

В XML-файл обычно включаются определения параметров (или вариантов выбора), которые могут быть заданы для данного экземпляра типа пункта меню. Каждый такой параметр размечается отдельным полем в элементе `fieldset`, который, в свою очередь, размещается в элементе разметки `fields`. Эти параметры разделяются на два возможных типа элемента разметки `fields`: `request` и `params`.

Элемент разметки `fields` типа `request`

Первый элемент `fields` из XML-файла, определяющего разметку типа пункта меню для выбора единственной категории веб-ссылок, выглядит следующим образом:

```
<!-- добавить поля в переменные запроса для компоновки -- >
<fields name="request">
  <fieldset name="request">

    <field name="id" type="category"
        default="0"
        description="COM_WEBLINKS_FIELD_SELECT_CATEGORY_DESC"
        extension="com_weblinks"
        label="COM_WEBLINKS_FIELD_SELECT_CATEGORY_LABEL"
        required="true"
    />
  </fieldset>
</fields>
```

Значения параметров должны храниться где-то в базе данных. Так, для пунктов меню используется таблица `#__menus`, где эти значения сохраняются в столбце `link` или `params`. Обратите внимание на то, что атрибут `name` элемента разметки `fields` имеет значение `"request"`. Если в разметке имеется элемент `fields` с именем, обозначающим

запрос, то параметр сохраняется в столбце `link` данной таблицы как запрос из URL, используя следующий формат:

```
<имя> + = + <значение>
```

В приведенном выше примере такой параметр будет выглядеть следующим образом:

```
id=xx
```

где `xx` — идентификатор категории. При загрузке URL для данного пункта меню этот параметр включается в URL и становится доступным в переменной `$_REQUEST`. Как правило, элементы разметки `fields` типа `request` служат для хранения идентификатора главной строки в базе данных, используемой для хранения пункта меню. Например, во многих пунктах меню используются поля идентификаторов категорий, как показано выше, или других элементов содержимого, в том числе статей или контактной информации. В данном примере URL будет указывать на единственную категорию подписок, а идентификатор этой категории может быть получен из запроса. В некоторых случаях, в том числе в представлении `Featured Article` (Избранные статьи), идентификатор единственного элемента или категории отсутствует, и поэтому элемент разметки `fieldset` типа `request` не требуется.

Как и все остальные параметры вообще, рассматриваемые здесь параметры отображаются на экране в том порядке, в каком они располагаются в XML-файле, будучи сгруппированными в элементе разметки `fieldset`. Каждому элементу разметки `fieldset` соответствует свой слайдер, разворачивающийся или сворачивающийся по мере надобности.

Элемент разметки `fields` типа `params`

Второй элемент разметки `fields` из XML-файла начинается со следующего кода:

```
<!-- добавить поля в объект параметров для компоновки -- >
<fields name="params">
<fieldset name="basic" label="JGLOBAL_CATEGORY_OPTIONS">
    <field name="spacer1" type="spacer" class="text"
      label="JGLOBAL_SUBSLIDER_DRILL_CATEGORIES_LABEL"
    />

    <field name="show_category_title" type="list"
      label="JGLOBAL_SHOW_CATEGORY_TITLE"
      description="JGLOBAL_SHOW_CATEGORY_TITLE_DESC"
    >

      <option value="">JGLOBAL_USE_GLOBAL</option>
      <option value="0">JHIDE</option>
      <option value="1">JSHOW</option>
    </field>
```

Атрибут `name` этого элемента разметки `fields` имеет значение `"params"`. В силу этого все элементы, находящиеся внутри данного элемента разметки `fields`, будут сохраняться как символьная строка формата JSON в столбце `params` таблицы `#__menu`. Напомним, что в класс `JTable` включен метод `getItem()` для распаковки содержимого этого столбца в объект типа `JRegistry`, а также метод `bind()` для упаковки данного объекта обратно в символьную строку формата JSON. Элемент разметки `fields` может содержать

любое число вложенных в него элементов разметки `fieldset` и `field`. И в этом случае размечаемые с их помощью параметры отображаются на экране в том порядке, в каком они располагаются в XML-файле, будучи сгруппированными в слайдеры в элементе разметки `fieldset`.

Следует иметь в виду, что у пунктов меню имеются также стандартные, обычно отображаемые параметры. К их числу относятся следующие стандартные параметры: Link Type Options (Параметры типа ссылки), Page Display Options (Параметры отображения страницы), а также Metadata Options (Параметры метаданных). Все они извлекаются из файла `administrator/components/com_menus/models/forms/item_component.xml` и отображаются в форме после параметров из XML-файла компоновки. Но для некоторых типов пунктов меню, в том числе `External URL` (Внешний унифицированный указатель ресурса), `Menu Item Alias` (Псевдоним пункта меню) и `Text Separator` (Текстовый разделитель), все эти стандартные параметры не отображаются.

Маршрутизация в пользовательской части веб-сайта

Как упоминалось в главе 3, маршрутизатор представляет собой класс, преобразующий URL в массив команд. У этого класса имеются два открытых метода `parse()` и `build()`. Метод `parse()` принимает объект типа `JURI` и возвращает массив команд. А метод `build()` выполняет противоположную функцию, принимая массив команд и возвращая объект типа `JURI`.

Общие положения об удобных для поиска URL

В административной части веб-сайта эстетике оформления URL и способности поискового механизма найти по нему страницу обычно не уделяется никакого внимания. Поэтому URL

```
administrator/index.php?option=com_categories&extension=com_weblinks
```

оказывается вполне приемлемым, если по нему можно перейти туда, куда нужно. А в пользовательской части веб-сайта дело обстоит намного сложнее. Ведь именно здесь особое внимание уделяется внешнему виду URL, а также индексации страницы в поисковых механизмах, чтобы добиться хорошего их расположения в результатах поиска.

В первом поколении веб-сайты состояли из статических HTML-страниц, располагавшихся в отдельных HTML-файлах. А URL просто содержал имя домена и имя HTML-файла, например:

```
http://www.mydomain.com/homepage.html
```

или

```
http://www.mydomain.com/cars/sportscars.html
```

Когда же появились первые динамические веб-сайты, на них использовался формат запросов, аналогичный приведенному выше URL из административной части (например, `http://www.mydomain.com/index.php?option=com_category&view=blog&id=23`). Первоначально такие поисковые механизмы, как Google или Yahoo, не были вполне пригодны для индексирования страниц с символьными строками запросов в URL, и поэтому динамические страницы не получали такие же высокие места в результатах поиска, как и статические HTML-страницы. В связи с этим в Joomla и другие системы CMS была внедрена

возможность использовать удобные для поиска URL (так называемые SEF URL), которые были в большей степени похожи на старые URL статических HTML-страниц. Между тем, возможности поисковых механизмов индексировать страницы по URL с запросами постепенно усовершенствовались. И теперь поисковые механизмы, как правило, индексируют страницы как по URL с символьными строками запросов, так и по другим URL, а следовательно, на использование запросов в URL никаких ограничений больше не налагается.

Следовательно, термин *SEF URL* ныне употребляется не совсем верно. Основной причиной использования в настоящее время SEF URL, т.е. удобных для поиска URL, является стремление представить пользователю веб-сайта URL в более понятном для человека, удобочитаемом виде. Удобные для поиска URL уже стали нормой для большинства программных средств CMS. В Joomla администратор веб-сайта может сам выбирать, пользоваться ли удобными для поиска URL или нет. Кроме того, в целом ряде расширений Joomla предоставляются альтернативные способы создания удобных для поиска URL. Считается, что такие URL используются на подавляющем большинстве современных веб-сайтов.

Совсем другое дело, хотя и имеющее отношение к данной теме, — перезапись URL. Допустим, на популярном среди посетителей веб-сайте требуется заменить программное обеспечение, под управлением которого он работает, а некоторые его страницы занимают высокие места в результатах поиска в распространенных поисковых механизмах, и поэтому пользователи сделали на них закладки. Если новое программное обеспечение не поддерживает ту же самую структуру URL, оно может оказаться не в состоянии отобразить страницу по этому URL. В таком случае можно сопоставить старый URL с новым, предоставив тем самым возможность пользователям употреблять и дальше старый URL. В Joomla для этой цели можно воспользоваться компонентом переадресации или средствами перезаписи URL в программном обеспечении веб-сервера (например, модулем `mod_rewrite` на веб-сервере Apache).

Методика оптимального соответствия URL

Задача представления URL при отображении веб-страницы на первый взгляд кажется простой, но на самом деле она может оказаться много сложнее. Главное затруднение состоит в том, что во многих случаях страница, по существу, с той же самой информацией может быть отображена самыми разными способами. Для рассмотрения этого вопроса обратимся к примеру компонента Weblinks. Если загрузить выборочные данные (например, для версии Joomla 2.5), то появится категория `Sample Data-Weblinks` (Выборочные данные веб-ссылок) с идентификатором **18**. У нее имеется подчиненная категория `Joomla Specific Links` (Специальные ссылки Joomla) с идентификатором **32** и подчиненной ей самой категорией `Other Resources` (Другие ресурсы) с идентификатором **33**. В меню компонента Weblinks определены два пункта: один — для представления категорий (отображает категории, подчиненные категории `Sample Data-Weblinks`), другой — для представления одиночной категории. Пункт выбора одиночной категории определен для категории **32** (т.е. категории `Joomla Specific Links`).

Такая организационная структура допускает два способа поиска веб-ссылок для категории **32**, как показывают приведенные ниже примеры.

- Выбрать пункт Weblinks Single Category (Одиночная категория веб-ссылок) из меню Using Joomla (Применение Joomla).

- Выбрать пункт меню Weblinks Categories (Категории веб-ссылок) и щелкнуть на ссылке `Joomla Specific Links`.

Аналогичным образом можно отобразить ссылки на категорию **33**, щелкнув на ссылке `Other Resources`, находящейся на странице Weblinks Categories или на странице Weblinks Single Category. В подобных случаях приходится решать, какой именно URL следует отображать. В частности, URL может быть отображен по-разному, зависимости от того, каким образом пользователь перешел на данную страницу. В первом примере категорию `Using Joomla` можно было бы отобразить как ее собственный пункт меню, когда она выбрана из меню, но показать ее как подчиненную категорию пункта меню Categories, когда она выбрана из пункта меню категорий. Но при прочих равных условиях целесообразнее иметь тот же самый URL для одного и того же содержимого страницы. Такой подход оказывается более интуитивным и упрощает каталогизацию подобного содержимого в поисковых механизмах.

Именно по этой причине в стандартных маршрутизаторах Joomla применяется методика оптимального соответствия URL. Это означает, что система пытается найти существующий пункт меню для отображения данного содержимого. Если такой пункт меню имеется, можно всегда воспользоваться соответствующим ему URL при отображении страницы. В противном случае можно попытаться найти пункт меню, наиболее точно согласующийся с данным содержимым.

Рассмотрим действие этой методики на представленных ранее примерах. Если щелкнуть на категории `Joomla Specific Links`, находясь в представлении категорий, маршрутизатор осуществит поиск существующего пункта меню для данной категории веб-ссылок. Если этот пункт меню будет найден, то для него возвратится URL. В итоге отобразятся та же самая страница и URL, что и при выборе пункта Weblinks Single Category непосредственно из меню.

Во втором примере пункт меню для категории **33** (`Other Resources`) отсутствует. Из этого положения имеются два выхода. Во-первых, показать ее как подчиненную категорию пункта меню Weblinks Single Category, поскольку она подчинена категории `Joomla Specific Links`. И во-вторых, показать ее как подчиненную подкатегории `Sample Data-Weblinks`, выбираемой из пункта меню категорий. Если при определении маршрутизатора было решено, что представлению одиночной категории следует всегда отдавать приоритет над представлением категорий, то URL отображается как результат проникновения вглубь меню, начиная с пункта выбора одиночной категории. Иными словами, отображается URL для пункта меню одиночной категории, после которого следует идентификатор подчиненной категории и псевдоним. Это можно продемонстрировать на следующем проверочном примере. Если загружены стандартные выборочные данные, то при проникновении вглубь меню к категории `Other Resources` получается приведенный ниже URL.

```
index.php/using-joomla/extensions/components/weblinks-component/weblinks-
single-category/33-other-resources
```

В этом URL к пункту меню одиночной категории присоединяются идентификатор текущей категории (**33**) и псевдоним (`other-resources`). Тот же самый URL получается и при проникновении вглубь меню, начиная с пункта Weblinks Categories или Weblinks Single Category.

Если теперь перейти к административной части веб-сайта на Joomla и снять пункт меню Weblinks Single Category с публикации, а затем щелкнуть на категории Other Resources, выбрав ее на странице Weblinks Categories, то в конечном итоге будет получен следующий URL:

```
index.php/using-joomla/extensions/components/weblinks-component/weblinks-
categories/32-joomla-specific-links/33-other-resources
```

В связи с тем что единственная категория больше не доступна, пришлось выбрать следующий ближайший вариант, т.е. проникнуть вглубь меню категорий. Пункт меню категорий связан с категорией Sample Data-Weblinks, имеющей идентификатор **18**. Это прародительская категория для категории **33**. Таким образом, путь к текущей категории отображается, начиная с категории **18** и следуя дальше вниз по иерархии (в данном случае сначала к категории **32**, а затем к категории **33**).

А что, если соответствующие пункты меню отсутствуют? В таком случае категории и пункты меню одиночных категорий снимаются с публикации в административной части веб-сайта. Теперь пункты меню недоступны для выбора, но можно ввести вручную следующий URL в форме запроса, чтобы отобразить категорию веб-ссылок:

```
index.php?option=com_weblinks&view=category&id=33
```

По этому URL происходит переход к странице Other Resources, а запрос получается именно в той форме, в какой он был введен в URL. В итоге маршрутизатор получает возможность успешно загрузить компонент и представление, но ему не удастся найти сопоставимый удобный для поиска URL. Следует также заметить, что при загрузке страницы получаются только те модули, которые назначены для всех пунктов меню. И это происходит потому, что в запросе отсутствует идентификатор пункта меню (Itemid). Если бы он был введен в запрос, то были бы получены модули, шаблоны и средства языковой поддержки, назначенные для данного пункта меню.

Построение URL

А теперь рассмотрим более подробно, каким образом описанная выше методика реализуется непосредственно в коде. Начнем с файла views/categories/tmpl/default_items.php компоновки представления категорий. Этот файл отвечает за отображение информации по каждой категории веб-ссылок в отдельности. В нем находится следующий фрагмент кода:

```
<span class="item-title"><a href="<?php
echo JRoute::_(WeblinksHelperRoute::getCategoryRoute($item->id));?>">
    <?php echo $this->escape($item->title); ?></a>
</span>
```

В выделенной выше полужирным строке кода выполняются две операции. Сначала вызывается метод getCategoryRoute() с идентификатором категории ($item->id) в качестве аргумента. В результате создается ссылка в форме запроса из URL. Эта ссылка передается затем методу JRoute::_(), а результат его выполнения выводится по команде echo в виде URL в атрибуте href элемента привязки a.

Получение оптимально соответствующего пункта меню

В методе `getCategoryRoute()` из файла `helpers/route.php` обнаруживается пункт меню, оптимально соответствующий идентификатору текущей категории. Как же это происходит? Сначала проверяется наличие пункта меню единственной категории для данной категории. Если такой пункт меню имеется, то возвращается URL, указывающий на этот пункт меню, и все. А если соответствующий пункт меню отсутствует, то осуществляется поиск пункта меню веб-ссылок, указывающего на одну из родительских категорий (родительскую, прародительскую и т.д.) для текущей категории. Для этого создается массив всех ее родительских категорий, а затем каждая из них проверяется по списку пунктов меню веб-ссылок. Сначала проверяются представления единственных категорий, поскольку они подходят лучше, чем представления многих категорий. Если представление единственной категории не удастся найти для любой из родительских категорий, то проверяются пункты меню, соответствующие представлениям многих категорий. И наконец, если не удастся найти ни один из пунктов меню веб-ссылок или категорий, соответствующих любой из родительских категорий, то для создания ссылки просто выбирается активный в настоящий момент пункт меню.

А теперь перейдем непосредственно к коду. В первой части метода `getCategoryRoute()` для текущей категории получается объект типа `JCategoryNode`. Главной в этой части является следующая строка кода:

```
$category = JCategories::getInstance('Weblinks')->get($id);
```

А далее, после проверки категории на достоверность, следует приведенный ниже фрагмент кода.

```
        $needles = array(
        'category' => array($id)
    );

    if ($item = self::_findItem($needles)) {
      $link = 'index.php?Itemid='.$item;
    }
```

Именно здесь и делается попытка найти точное соответствие. Для этого сначала создается массив из представления категории и ее идентификатора, а затем проверяется, соответствует ли она одному из существующих пунктов меню, для чего вызывается метод `_findItem()`.

Обратите внимание на то, что для вызова метода `_findItem()` вместо выражения `$this->` используется префикс `self::`, поскольку этот метод является статическим. Ключевое слово `self` означает "этот класс", тогда как переменная `$this` ссылается на "этот объект". Таким образом, для вызова статических методов или обращения к статическим полям в том классе, где они определены, необходимо пользоваться ключевым словом `self`. Это ключевое слово было внедрено в версии 5 языка PHP.

Если обнаружено точное соответствие категории существующему пункту меню, то поставленная задача считается выполненной. В противном случае выполняется приведенный ниже фрагмент кода.

```
// создать ссылку
$link = 'index.php?option=com_weblinks&view=category&id='.$id;
```

```
if ($category) {
    $catids = array_reverse($category->getPath());
    $needles = array(
        'category' => $catids,
        'categories' => $catids
    );

    if ($item = self::_findItem($needles)) {
        $link .= '&Itemid='.$item;
    }
    else if ($item = self::_findItem()) {
        $link .= '&Itemid='.$item;
    }
}
```

В этом фрагменте кода сначала создается ссылка, кроме идентификатора категории. А затем составляется список идентификаторов родительских категорий для идентификатора текущей категории, для чего вызывается метод `getPath()` из класса `JCategoryNode`. Напомним, что в данном примере родительской для категории с идентификатором **33** (`Other Resources`) является категория с идентификатором `32` (`Joomla Specific Links`), а для нее родительской — категория с идентификатором **18** (`Sample Data-Weblinks`). Метод `getPath()` возвращает все родительские категории, обнаруженные в массиве, используя для этого формат <идентификатор> + <псевдоним>. Так, для категории с идентификатором **33** из этого метода возвращается следующий результат:

```
0 => 18:sample-data-weblinks
1 => 32:joomla-specific-links
2 => 33:other-resources
```

Содержимое массива просто обращается с помощью функции `array_reverse()`, чтобы идентификатор `33` стал нулевым элементом, а идентификатор **18** — третьим элементом данного массива, как показано ниже.

```
0 => 33:other-resources
1 => 32:joomla-specific-links
2 => 18:sample-data-weblinks
```

Полученный результат сохраняется в переменной `$catids`.

Затем создается массив `$needles`, предназначенный для сравнения по списку пунктов меню веб-ссылок. Этот массив является ассоциативным и состоит из двух элементов с ключами `category` и `categories`, значением для которых является массив `$catids`.

Далее метод `self::_findItem($needles)` вызывается еще раз. На этот раз поиск расширяется, включая любую родительскую категорию и пункт меню категорий. Первым ключом в ассоциативном массиве `$needles` является категория, и поэтому сначала среди всех пунктов меню веб-ссылок ищется пункт меню единственной категории, чтобы выяснить, соответствует ли идентификатор этой категории одному из идентификаторов родительских категорий. Если соответствие не будет найдено, то повторяется процесс проверки пунктов меню веб-ссылок с представлением, равным категориям.

Ниже приведен первый фрагмент кода из метода `_findItem()`.

```
protected static function _findItem($needles = null)
{
    $app = JFactory::getApplication();
    $menus = $app->getMenu('site');

    // подготовить массив к поиску в обратном порядке
    if (self::$lookup === null) {
      self::$lookup = array();

      $component = JComponentHelper::getComponent('com_weblinks');
      $items = $menus->getItems('component_id', $component->id);
      foreach ($items as $item)
      {
        if (isset($item->query) && isset($item->query['view'])) {
        $view = $item->query['view'];

        if (!isset(self::$lookup[$view])) {
          self::$lookup[$view] = array();
        }

        if (isset($item->query['id'])) {
          self::$lookup[$view][$item->query['id']] = $item->id;
        }
      }
    }
}
```

В переменной `$menus` хранится объект типа `JMenuSite`. В классе `WeblinksHelperRoute` имеется статическое поле `$lookup`. А поскольку оно статическое, то значение в нем достаточно установить лишь один раз. До тех пор пока этот класс находится в оперативной памяти, его статические поля остаются определенными. В первом условном операторе `if` из приведенного выше фрагмента кода проверяется, установлено ли данное поле. Если оно установлено, то делать этого больше не нужно.

Для организации массива `$items` из всех пунктов меню компонента Weblinks вызывается метод `getItems()` из класса `JMenuSite`. Этот метод возвращает массив объектов стандартных классов с полями для хранения отдельных пунктов меню веб-ссылок. Затем организуется циклическое обращение к пунктам меню веб-ссылок в данном массиве и создается ассоциативный массив `$lookup`, ключом к которому служит преставление, извлекаемое из запроса пункта меню, а значением — еще один ассоциативный массив. Ключом к этому массиву служит извлекаемый из запроса идентификатор (т.е. идентификатор категории), а значением — идентификатор пункта меню. В рассматриваемом здесь примере с загруженными выборочными данными содержимое массива `$lookup` определяется следующим образом:

```
categories => array (0 => 438, 18 => 227)
category => array(32 => 274, 31 => 296)
```

Следует заметить, что конкретные идентификаторы могут отличаться в зависимости от изменений в выборочных данных, но описанный выше принцип остается неизменным. Как видите, в данном массиве определены четыре пункта меню веб-ссылок: два — для выбора предствления многих категорий и еще два — для выбора предствления единственной

категории. Пункты меню многих категорий обозначаются значением 438, указывающим на категорию с идентификатором **0** (т.е. корневую категорию для всей таблицы), а значение 227 указывает на категорию с идентификатором **18** (Sample Data-Weblinks). Пункты меню единственных категорий обозначаются значением **274**, указывающим на категорию с идентификатором **32**, а значение **296** указывает на категорию с идентификатором **31**.

После того как массив $lookup будет определен, можно попытаться найти оптимальное соответствие для текущей категории, используя массив $needles. Исходный код, необходимый для этой цели, приведен ниже.

```
if ($needles) {
  foreach ($needles as $view => $ids)
  {
    if (isset(self::$lookup[$view])) {
      foreach($ids as $id)
      {
        if (isset(self::$lookup[$view][(int)$id])) {
        return self::$lookup[$view][(int)$id];
        }
      }
    }
  }
}
```

Циклическое обращение к массиву $needles производится по искомому представлению. Сначала в нем ищется представление единственной категории, а затем представление многих категорий. Поочередно просматривается каждый идентификатор $ids в иерархии категорий. Напомним, что порядок следования элементов данного массива изменен на обратный, и поэтому текущая категория (с идентификатором **33**) располагается в нем первой, затем следует родительская категория (с идентификатором **32**), далее — прародительская категория (с идентификатором **18**) и т.д. Следовательно, поиск происходит в установленном порядке предпочтений. И как только будет найдено точное соответствие представления идентификатору категории (в выделенной выше полужирным строке кода), возвращается идентификатор соответствующего пункта меню, который в данном случае является значением, извлекаемым из элемента массива $lookup. После выполнения команды return происходит немедленный возврат из данного метода.

Напомним, что среди прочих значений в массиве $needles находится псевдоним категории (например, "33:other resources"), а следовательно, значение переменной $id будет представлено в таком формате. Именно поэтому в рассматриваемом здесь коде используется оператор приведения (int), чтобы преобразовать значение переменной $id в идентификатор категории (например, **33**) при обращении к массиву $lookup.

А теперь осталось лишь рассмотреть случай, когда соответствие не найдено. В этом случае просто возвращается идентификатор активного в текущий момент пункта меню. После возврата к методу getCategoryRoute() идентификатор пункта меню присоединяется к ссылке (например, "&Itemid=274"), которая и возвращается. В итоге получается ссылка, указывающая на пункт меню веб-сайта, в наибольшей степени соответствующий идентификатору текущей категории. Им оказывается один из следующих пунктов меню в порядке очередности.

- Пункт меню единичной категории веб-ссылок для выбора данной категории.
- Пункт меню единичной категории веб-ссылок для выбора одной из родительских категорий данной категории.
- Пункт меню многих категорий веб-ссылок для выбора одной из родительских категорий данной категории.
- Активный в текущий момент пункт меню.

Поиск пути к категории из пункта меню

А теперь, когда получен идентификатор пункта меню (`Itemid`), можно завершить процесс маршрутизации. Напомним, что он был начат в следующей строке кода:

```
echo JRoute::_(WeblinksHelperRoute::getCategoryRoute($item->id));
```

Теперь упомянутая выше ссылка передается в качестве аргумента методу `JRoute::_()`. В этом методе сначала получается маршрутизатор, который в пользовательской части веб-сайта определяется в классе `JRouterSite` из файла `includes/router.php`, а затем вызывается метод `build()` из класса `JRouterSite`. Если установлена кнопка-переключатель Yes параметра Search Engine Friendly URLs на вкладке SEO диалогового окна Global Configuration, то метод `build()` принимает в качестве аргумента URL с запросом (например, `index.php?option=com_weblinks&Itemid=274`) и возвращает удобный для поиска URL (например, `index.php/using-joomla/extensions/components/weblinks-component/weblinks-single-category`).

Прежде всего в методе `build()` вызывается другой метод `build()` из его родительского класса `JRouter`. Если установлена кнопка-переключатель Yes упомянутого выше параметра Search Engine Friendly URLs, то далее вызывается защищенный метод `_buildSefRoute()` (снова из класса `JRouterSite`). В этом методе имеется следующий фрагмент кода:

```
// использовать обработчик маршрутизации для компонента,
// если таковой имеется
$path = JPATH_SITE . '/components/' . $component . '/router.php';

// использовать специальный обработчик маршрутизации,
// если таковой имеется
if (file_exists($path) && !empty($query)) {
    require_once $path;
    $function = substr($component, 4).'BuildRoute';
    $function = str_replace(array("-", "."), "", $function);
    $parts = $function($query);
```

В этом фрагменте кода проверяется наличие маршрутизатора, предоставляемого для компонента, в соответствии с принятыми условными обозначениями имен. Такой маршрутизатор должен находиться в файле `router.php` на верхнем уровне структуры файлов данного компонента. В данном файле должна присутствовать функция со следующим названием:

```
<имя компонента (за вычетом части "com_")> + "BuildRoute"
```

Если такой файл найден, то данная функция вызывается в последней строке приведенного выше фрагмента кода и возвращает массив `$parts`. Для компонента Weblinks такой

файл имеется и называется components/com_weblinks/router.php, а вызываемая из него функция — WeblinksBuildRoute(). Следует иметь в виду, что в этом файле объявляются только методы build() и parse(), но не класс.

Как пояснялось ранее, если имеется пункт меню единственной категории, указывающий на эту категорию веб-ссылок, маршрутизация осуществляется непосредственно к этому пункту меню. Следовательно, функция WeblinksBuildRoute() возвращает пустой массив, если текущая категория совпадает с категорией, указываемом в пункте меню единственной категории.

С другой стороны, для перехода к категории веб-ссылок, с которой не связан пункт меню, приходится сначала выяснять, какая из родительских категорий, если таковые имеются, соответствует оптимально выбираемому пункту меню, а затем показывать часть иерархии категорий, связывающую категорию данного пункта меню с текущей категорией. И все это делается в приведенном ниже фрагменте кода.

```
if ($category) {
  // Что сделать: сгенерировать ошибку, если категория не существует
  // или снята с публикации
  $path = $category->getPath();
  $path = array_reverse($path);

  $array = array();
  foreach($path as $id)
  {
    if ((int) $id == (int)$menuCatid) {
      break;
    }

    if ($advanced) {
      list($tmp, $id) = explode(':', $id, 2);
    }

    $array[] = $id;
  }
  $segments = array_merge($segments, array_reverse($array));
}
```

Этот фрагмент кода требует подробного рассмотрения, поскольку он не так прост. На данный момент уже известно, что имеется категория, для которой отсутствует соответствующий пункт меню. Но у одной из ее родительских категорий, вероятно, такой пункт меню имеется. Это известно из того, что ссылка на такую категорию уже содержит оптимально соответствующий пункт меню.

В переменной $category содержится объект типа JCategoryNode для текущей категории. И в этом случае метод getPath() вызывается, чтобы возвратить всю иерархию для данной категории. Затем порядок следования элементов в массиве изменяется на обратный с помощью функции array_reverse(), как было показано в предыдущем примере.

Далее организуется циклическое обращение к родительским категориям. Ключом к пониманию этой операции служит код, выделенный выше полужирным. Если искомое соответствие не найдено, значение переменной $id вводится в массив $array и далее цикл повторяется. Когда же соответствие наконец обнаружено, выполнение цикла foreach прерывается по команде break и происходит немедленный выход из него.

В конечном итоге текущая категория и все ее родительские категории вверх по иерархии до совпадающей категории, но не включая ее, вводятся в массив. Затем порядок следования элементов в данном массиве меняется на обратный, чтобы расположить все обнаруженные категории по иерархии сверху вниз.

И наконец, вызывается функция `array_merge()`, чтобы ввести полученный результат в массив `$segments`, который и возвращается в виде значения в конце рассматриваемого здесь метода. Если обратиться к примеру проникновения вглубь меню до категории `Other Resources`, то массив `$segments` будет содержать один приведенный ниже элемент.

```
0 => 33:other-resources
```

В данном примере пункт меню Weblinks Single Category снят с публикации, и поэтому оптимально соответствующим оказывается пункт меню Weblinks Categories. Следовательно, массив `$segments` будет содержать два приведенных ниже элемента.

```
0 => 33:other-resources, 1 => 32:joomla-specific-links
```

Далее происходит возврат в вызывающий метод, которым в данный момент является метод `_buildSefRoute()` из класса `JRouterSite`. В этом методе отдельные части URL собираются вместе и разделяются знаками косой черты (/), а в самом начале к ним добавляется префикс `component/`. И наконец, по полученному в итоге значению задается путь для объекта типа `JURI`.

Итак, URL, указывающий на наиболее точно соответствующий пункт меню, создан. По мере надобности в него был также включен путь от категории данного пункта меню вниз по иерархии к текущей категории. В итоге получился логически согласованный и удобный для пользователей URL.

Синтаксический анализ URL

Процесс построения URL был начат с идентификатора категории веб-ссылок и доведен до представления текущей категории. Но, помимо этого, требуется организовать процесс и в обратном направлении, т.е. взять удобный для поиска URL, построенный с помощью метода `build()`, и получить из него запрос, который приведет к нужному представлению.

Этот процесс аналогичен описанной ранее методике построения URL. Начинается он с вызова функции `$app->route()` из файла `index.php` на верхнем уровне структуры файлов данного компонента. В итоге получается объект типа `JRouterSite` и выполняется его метод `parse()`. Как и в предыдущем примере, в этом методе вызывается другой метод `parse()` из его родительского класса, который, в свою очередь, вызывает метод `JRouterSite_parseSefRoute()`. В этом методе обнаруживается та часть удобного для поиска URL, в которой указывается пункт меню (например, `components/weblinks-component/single-weblink-category`), и на ее основании задается конкретный пункт меню. Затем часть URL, указывающая на пункт меню, отбрасывается и оставляется только путь к категории, который преобразуется в массив элементов, разделяемых знаками косой черты (/), причем каждая его часть указывается в следующем формате: `<идентификатор> + : + <псевдоним>`.

Далее вызывается метод `parse()` из компонента — в данном случае это метод `WeblinksParseRoute()`, которому в качестве аргумента передается сформированный ранее массив значений пути к категории. В этом методе получаются категории, подчиненные

категории данного пункта меню. Поле заготовки каждой подчиненной категории, указываемое в том же формате, что и отдельные части пути к данной категории, сравнивается с категориями из переданного массива. Как только будет найдено соответствие, создается ассоциативный массив с идентификатором категории и представлением категории. Этот массив возвращается вызывающему методу, а его элементы вводятся в запрос из URL. В данном примере удобный для поиска URL преобразуется в следующий массив:

```
Itemid => 274, option => com_weblinks, id => 33, view => category
```

Именно эта информация и требуется в Joomla для правильной загрузки компонента и его представления.

Пользовательское представление ленты новостей

К числу самых примечательных отличий пользовательской части компонента Weblinks от административной его части относится представление ленты новостей. В базовом компоненте Weblinks имеется возможность реализовать ленту новостей в представлении единственной категории. Лента новостей позволяет подписаться на данный пункт меню и автоматически получать обновления по мере смены страницы.

Такая возможность встроена в Joomla, а следовательно, ввести ленту новостей в любой компонент не составит особого труда. Напомним, что в экранной форме Edit Menu Item (Редактирование пункта меню) имеется возможность отображать ссылку на ленту подачи информации. Если проанализировать исходный код метода `display()` из файла `views/category/view.html.php`, то можно обнаружить, что в нем вызывается метод `_prepareDocument()`. В конце этого метода имеется следующий фрагмент кода:

```
// добавить альтернативную ссылку на ленту подачи информации
if ($this->params->get('show_feed_link', 1) == 1)
{
    $link = '&format=feed&limitstart=';
    $attribs = array('type' => 'application/rss+xml', 'title' =>
↳'RSS 2.0');
    $this->document->addHeadLink(JRoute::_($link.'&type=rss'),
↳'alternate', 'rel', $attribs);
    $attribs = array('type' => 'application/atom+xml', 'title' =>
↳'Atom 1.0');
    $this->document->addHeadLink(JRoute::_($link.'&type=atom'),
↳'alternate', 'rel', $attribs);
}
```

Если для данного элемента выбрать отображение ссылки на ленту подачи информации, то в конечном итоге будут созданы две ссылки: одна — для подачи информации в формате RSS 2.0, а другая — в формате Atom 1.0. Для маршрутизации, как и прежде, используется метод `JRoute::_()`, а сама ссылка указывается в качестве аргумента метода `addHeadLink()` из класса `JDocument`. В итоге создается ссылка, требующаяся пользователю для подписки на ленту подачи информации. Например, в браузере Firefox эта ссылка отображается в виде пиктограммы с распространяющимися в эфир радиоволнами слева от закладки "Лента новостей".

Следует заметить, что в данной ссылке формат указывается следующим образом: `format=feed`. При загрузке URL это значение вводится в запрос. Когда для этого URL

создается объект документа, то вместо обычного объекта типа JDocumentHTML фактически создается объект типа JDocumentFeed. В методе display() из класса JController переменной $viewType присваивается значение "feed". А при выполнении метода getView() вместо файла view.html.php загружается файл views/category/view.feed.php.

По своему содержимому файл view.feed.php похож на файл view.html.php. Прежде всего следует заметить, что в нем объявляется тот же самый класс WeblinksViewCategory, что и в файле view.html.php, только информация в нем представлена в другом формате, и поэтому у этого класса такое же самое имя. Но в любом цикле запроса загружается только один из этих классов. У данного класса имеется единственный метод display() с очень простым исходным кодом. Сначала в нем получаются объекты приложения и документа и устанавливается URL для ссылки на ленту подачи информации с помощью методов JRoute::_() и WeblinksHelperRoute::getCategoryRoute(), а затем устанавливаются некоторые значения для представления, в том числе количество отображаемых элементов, адрес электронной почты веб-сайта и сведения для редактора. Все эти установки делаются в экранной форме Global Configuration.

Далее из модели получаются требующиеся элементы и их категория. К этим элементам происходит поочередное обращение в цикле, и для каждого из них формируется отдельный URL. Кроме того, из заглавия элемента удаляется любой присутствующий в нем код HTML, а дата преобразуется в строковый формат. После этого создается новый объект типа JFeedItem и в него вводится новая информация. И наконец объект типа JFeedItem вводится в документ с помощью метода addItem() из класса JDocumentFeed.

Вот, собственно, и все. Все специальное форматирование, требующееся для лент новостей, выполняется автоматически в классах JDocumentFeed и JFeedItem. Этим же способом ленту новостей можно добавить в любой другой компонент.

Резюме

В этой главе было завершено рассмотрение базового компонента Weblinks. В начале главы были представлены классы представлений из административной части данного компонента. Попутно была рассмотрена компоновка экранной формы диспетчера веб-ссылок и применение класса JForm в экранной форме редактирования веб-ссылок. Затем обсуждение перешло к пользовательской части данного компонента, чтобы сравнить ее сходства и отличия от административной части. После этого были подробно рассмотрены особенности построения и синтаксического анализа удобных для поиска URL. И в заключение главы было показано, каким образом формируются ленты новостей.

В двух последующих главах полученные знания и навыки можно будет применить на практике для создания собственного компонента.

9

Компоненты, часть III. Пример разработки административной части компонента

В двух предыдущих главах на примере компонента Weblinks было показано, каким образом компоненты реализуются в Joomla по шаблону проектирования MVC (Модель–Контроллер–Представление). А в этой главе полученные знания и навыки будут применены для разработки административной части создаваемого примера компонента. Этот компонент мы попытаемся сделать как можно более простым, воспользовавшись в то же время большинством наиболее важных для этой цели средств из тех, что были рассмотрены ранее.

Примечание

Исходный код рассматриваемого здесь примера компонента можно загрузить по адресу `http://www.joomlaprogrammingbook.com`. Вы можете загрузить и установить исходный код перед тем, как читать эту главу, или же создавать данный пример компонента, постепенно прорабатывая ее материал и пользуясь загруженным кодом только для проверки своей работы. Второй способ потребует больших усилий, но в то же время позволит лучше понять исходный код.

Краткий обзор назначения создаваемого примера компонента

Сделаем прежде всего краткий обзор функционального назначения создаваемого компонента. Назовем его JoomPro Subscriptions. Его основное назначение — дать администратору веб-сайта возможность составлять подписки, а пользователям — подписываться на них. У каждой подписки должно быть свое описание, категория

и срок действия (в днях). Кроме того, каждая подписка должна быть связана с обычной группой пользователей веб-сайта на Joomla, задаваемой по команде меню Users⇨Groups (Пользователи⇨Группы). Когда пользователь подписывается на конкретную подписку, он автоматически вводится в группу. Кроме того, начальная и конечная даты подписки данного пользователя должны храниться в таблице базы данных.

Данный пример не охватывает то, что происходит после подписки пользователей. Принимая во внимание тот факт, что пользователи вводятся в группу, можно, в частности, предположить, что они получают доступ к ограниченной части веб-сайта. Кроме того, от подписывающихся пользователей потребуется лишь одно: согласиться с условиями обслуживания. Уяснив этот пример, вам будет нетрудно понять, каким образом дополнить рассматриваемый здесь компонент новыми свойствами, например, сделать подписку платной.

Практически весь исходный код рассматриваемого здесь примера компонента будет представлен в листингах этой главы. Как упоминалось выше, весь исходный код данного компонента можно загрузить в архивном виде с веб-сайта, посвященного этой книге, по адресу `http://www.joomlaprogrammingbook.com`.

Детальная разработка компонента

Для реализации проектного решения нам потребуются экранные формы диспетчеров, с помощью которых можно будет управлять подписками и их категориями в административной части компонента. Нам потребуется также экранная форма для ввода и редактирования подписок с полями названия, описания, даты публикации, идентификатора категории, идентификатора группы пользователей, срока действия (в днях), а также идентификатора доступа. А для указания первичного ключа к подписке будет использовано поле с целочисленным идентификатором. Кроме того, нам потребуется таблица для хранения информации о подписках каждого пользователя. По каждой подписке в этой таблице должны храниться идентификатор пользователя, идентификатор подписки, а также начальная и конечная даты подписки.

В пользовательской части создаваемого компонента должно быть представление подписок по категориям. Если пользователь получает разрешение редактировать подписку (с помощью стандартной системы списков контроля доступа — ACL), он может щелкнуть на этой подписке и подписаться на нее. И если подписка завершится успешно, должен появиться экран с благодарностями пользователю за сделанную подписку.

Итак, подведем краткий итог разработке данного компонента с точки зрения административных и пользовательских представлений. В административной части создаваемого компонента должно быть следующее.

- Экран Subscription Manager: Subscriptions (Диспетчер подписок: подписки; называется `submanager`).
- Экран Subscription Manager: Categories (Диспетчер подписок: категории; управляется компонентом `com_categories`).
- Экран Subscription Manager: Add/Edit (Диспетчер подписок: Ввод/Редактирование; называется `subscription`).

В пользовательской части создаваемого компонента должно быть следующее.

- Представление для отображения подписок по представлению категории (называется `category`).
- Представление для совершения подписки (называется `form`).

Кроме того, нам потребуются две таблицы в базе данных.

- Таблица для хранения списка доступных подписок (называется `#__joompro_subscriptions`).
- Таблица для хранения сведений о подписках каждого пользователя (называется `#__joompro_sub_mapping`).

Приведенных выше исходных данных должно быть достаточно для того, чтобы приступить к написанию кода. Но следует заметить, что не существует какого-то "правильного" порядка программирования компонентов. Этот процесс можно, в частности, начать с таблиц базы данных, а затем перейти к моделям, контроллерам и представлениям. В данном примере мы последуем за ходом выполнения программы, рассмотрев сначала программные средства, вызываемые при пользовании экраном диспетчера и затем при вводе и редактировании информации в административной части. А в следующей главе перейдем к пользовательской части создаваемого компонента, рассмотрев программные средства для составления списка категорий и подписки.

Файлы из административной части компонента

В табл. 9.1 перечислены файлы, требующиеся в административной части создаваемого компонента. Среди них отсутствует файл `index.html`, который должен непременно быть в каждой папке компонента. Пути ко всем этим файлам указаны относительно исходной папки `administrator/components/com_joomprosubs`.

Таблица 9.1. Файлы из административной части компонента подписок

Имя файла	Содержимое
`access.xml`	Действия системы списков контроля доступа (ACL) на уровне компонента и категории
`config.xml`	Поля для ввода параметров компонента
`controller.php`	Класс `JoomproSubsController`
`joomprosubs.php`	Сценарий, служащий в качестве точки входа в компонент
`joomprosubs.xml`	Установочный XML-файл
`controllers/subscription.php`	Класс `JoomprosubsControllerSubscription` (контроллер ввода и редактирования единственного элемента)
`controllers/submanager.php`	Класс `JoomprosubsControllerSubManager` (контроллер экрана диспетчера)
`helpers/joomprosubs.php`	Класс `JoomproSubsHelper`

Окончание табл. 9.1

Имя файла	Содержимое
language/en-GB/en-GB.com_joomprosubs.ini	Основной файл языковой поддержки компонента
language/en-GB/en-GB.com_joomprosubs.sys.ini	Системный файл языковой поддержки компонента
models/subscription.php	Класс JoomprosubsModelSubscription (модель ввода и редактирования единственного элемента)
models/submanager.php	Класс JoomprosubsModelSubManager (модель экрана диспетчера)
models/forms/subscription.xml	XML-файл разметки экранной формы ввода и редактирования, определяемой классом JForm
sql/install.mysql.utf8.sql	Сценарий SQL для установки базы данных
sql/uninstall.mysql.utf8.sql	Сценарий SQL для удаления базы данных
tables/subscription.php	Класс JoomproSubsTableSubscription (для записи строк в таблицу базы данных)
views/subscription/view.html.php	Класс JoomprosSubsViewSubscription (представление единственного элемента)
views/subscription/tmpl/edit.php	Файл компоновки экранной формы ввода и редактирования единственного элемента
views/submanager/view.html.php	Класс JoomprosSubsViewSubManager (представление экрана диспетчера)
views/submanager/tmpl/default.php	Файл компоновки экрана диспетчера

В административной части компонента прежде всего необходимо создать папку administrator/components/com_joomprosubs. Все файлы из административной части компонента должны находиться в этой папке, и поэтому здесь и далее они будут указываться относительно этой исходной папки.

В папке административной части компонента будет создана точка входа в эту часть компонента в виде файла joomprosubs.php. Напомним, что этот файл автоматически загружается в Joomla, если в URL для перехода к административной части делается ссылка option=com_joomprosubs.

Ниже приведен исходный код из файла joomprosubs.php.

```
<?php
/**
 * @package      Joomla.Site
 * @subpackage   com_joomprosubs
 * @copyright    Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↪reserved.
 * @license      GNU General Public License version 2 or later; see LICENSE.
txt
*/

// прямой доступ отсутствует
defined('_JEXEC') or die;

// проверить права доступа
```

```
if (!JFactory::getUser()->authorise('core.manage', 'com_joomprosubs')) {
    return JError::raiseWarning(404, JText::_('JERROR_ALERTNOAUTHOR'));
}

// включить зависимости
jimport('joomla.application.component.controller');

$controller = JController::getInstance('JoomproSubs');
$controller->execute(JRequest::getCmd('task'));
$controller->redirect();
```

Этот код должен быть вам уже знаком. Он аналогичен исходному коду из файла `administrator/components/com_weblinks/weblinks.php`, рассматривавшегося в главе 7, за исключением того, что в нем проверяются полномочия пользователя на доступ к компоненту `com_joomprosubs`, а также получается контроллер для этого компонента (по значению `"JoomproSubs"`).

Экран Subscriptions Manager: Subscriptions

На этом экране отображается список тех подписок, которые были созданы администратором веб-сайта. На рис. 9.1 приведен моментальный снимок этого экрана. Он похож на экран Weblinks Manager и другие экраны диспетчеров базовых компонентов Joomla. Следует заметить, что на этом экране отображаются все средства, характерные для экранов диспетчеров базовых компонентов Joomla. К их числу относятся следующие:

- панель инструментов для выбора типичных задач (создания, редактирования и прочих);
- поле ввода фильтров для отбора подписок по названию;
- раскрывающиеся списки для фильтрации подписок по состоянию "опубликовано", категории и уровню доступа;
- флажки, позволяющие панелям инструментов работать со многими элементами;
- столбцы с сортируемыми по порядку заголовками;
- элементы управления разбиением на страницы.

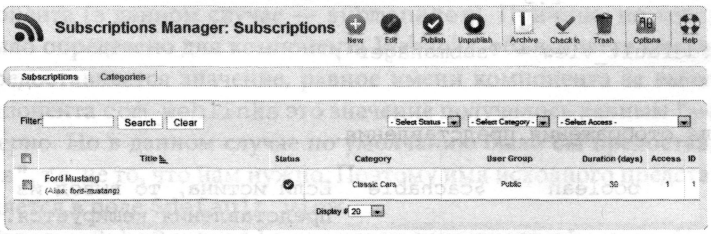


Рис. 9.1. Экран Subscriptions Manager: Subscriptions

Для создания данного экрана используются следующие программные средства:

- файл `controller.php` (класс `JoomproSubsController`);
- файл `views/submanager/view.html.php` (класс `JoomproSubsViewSubManager`);
- файл `helpers/joomprosubs.php` (класс `JoomproSubsHelper`);

- файл `models/submanager.php` (класс `JoomproSubsModelSubManager`);
- файл `sql/install.mysql.utf8.sql`;
- файл `views/submanager/tmpl/default.php`.

Исходный контроллер

Как пояснялось в главе 7, когда загружается экран диспетчера компонента, первым загружается класс исходного контроллера с выполняемой по умолчанию задачей отображения. В данном случае исходным является контроллер из класса `JoomproSubsController`. Исходный код этого класса приведен в листинге 9.1.

Листинг 9.1. Класс `JoomproSubsController` из файла `controller.php`

```
<?php
/**
 * @package       Joomla.Administrator
 * @subpackage    com_joomprosubs
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↳reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.
txt
*/

// прямой доступ отсутствует
defined('_JEXEC') or die;

/**
 * Контроллер Joomprosubs joomprosub
 *
 * @package       Joomla.Administrator
 * @subpackage    com_joomprosubs
*/
class JoomproSubsController extends JController
{
  /**
   * @var string        Исходное представление
   * @since             2.5
  */
  protected $default_view = 'submanager';

  /**
   * Метод для отображения представления.
   *
   * @param       boolean     $cachable   Если истина, то вывод из
   *                                      представления кешируется.
   * @param       array $urlparams        Массив параметров безопасного
   *                                      URL и типов их переменных,
   *                                      а действительные значения см.
   *                                      в {@link JFilterInput::clean()}.
   *
   * @return            JController       Этот объект поддерживает
   *                                      связывание методов в цепочку.
   */
```

```
  public function display($cachable = false, $urlparams = false)
  {
    JLoader::register('JoomproSubsHelper',
↳JPATH_COMPONENT.'/helpers/joomprosubs.php');

    // загрузить подменю
    JoomproSubsHelper::addSubmenu(JRequest::getCmd('view', 'submanager'));

    $view = JRequest::getCmd('view', 'submanager');
    $layout = JRequest::getCmd('layout', 'default');
    $id = JRequest::getInt('id');

    // проверить наличие формы редактирования
    if ($view == 'subscription' && $layout == 'edit' &&
↳!$this->checkEditId('com_joomprosubs.edit.subscription', $id)) {
        // не разрешать несанкционированный доступ к форме
        $this->setError(JText::sprintf(
↳'JLIB_APPLICATION_ERROR_UNHELD_ID', $id));
        $this->setMessage($this->getError(), 'error');
        $this->setRedirect(JRoute::_(
↳'index.php?option=com_joomprosubs&view=submanager', false));
        return false;
    }

    parent::display();

    return $this;
  }
}
```

Приведенный выше исходный код класса `JoomproSubsController` мало чем отличается от исходного кода класса `WeblinksController`, рассматривавшегося в главе 7. Этот класс расширяет класс `JController`. Обратите внимание на то, что в нем введено защищенное поле `$default_view`, в котором установлено значение `"submanager"`. В аналогичном контроллере компонента Weblinks это поле не было определено, но именно в нем указывается имя исходного представления для административной части создаваемого компонента (в данном случае — `submanager`). Причина, по которой аналогичное поле не было определено для компонента Weblinks, объясняется тем, что в Joomla по умолчанию предоставляется значение, равное имени компонента за вычетом префикса `com_`. Для компонента `com_weblinks` это значение получалось равным `"weblinks"`, что совершенно верно. Но в данном случае по умолчанию было бы предоставлено значение `"joomprosubs"`, т.е. не то, что нам нужно. Поэтому имя исходного представления специально определяется в поле `$default_view`.

У класса `JoomproSubsController` имеется единственный метод `display()`, в котором включается вспомогательный файл. Следует заметить, что для этой цели вместо команды `require_once` вызывается метод `JLoader::register()`. Напомним, что этот метод выполняется быстрее и считается более предпочтительным для загрузки класса. Для указания пути к папке компонента (в данном случае — `administrator/components/com_joomprosubs`) служит константа `JPATH_COMPONENT`, а для обращения к вспомогательному классу вызывается метод `addSubmenu()`.

Затем из запроса получается представление, компоновка, а также идентификатор подписки. Попутно поверяется, не попытался ли пользователь перейти к форме редактирования непосредственно. И наконец, вызывается метод `display()` из родительского класса `JController`, чтобы получить требуемое представление и выполнить его.

Контроллер подчиненного диспетчера и задачи на панели инструментов

Исходный контроллер обрабатывает задачу отображения, которая частично состоит в отображении панели инструментов, как показано на рис. 9.1. Первыми двумя на этой панели оказываются пиктограммы задач создания (New) и редактирования (Edit). Обе эти задачи обрабатываются контроллером подписки, рассматриваемым далее в главе. Остальные задачи, доступные на панели инструментов, выполняют публикацию, снятие с публикации, архивирование, регистрацию, удаление в корзину, настройку параметров и получение справки.

Как пояснялось в главе 7, задачи публикации, снятия с публикации, архивирования и удаления в корзину сопоставляются в контроллере с методом `publish()`, тогда как задача регистрации — с методом `checkin()`. Напомним также, что у класса `WeblinksControllerWeblinks` имеется единственный метод `getModel()`, поскольку он наследует методы `publish()` и `checkin()` из своего родительского класса `JControllerAdmin`. Это же относится и к классу контроллера `JoomproSubsControllerSubManager` из файла `controllers/submanager.php`. Поэтому ему требуется единственный метод `getModel()`, как следует из листинга 9.2.

Листинг 9.2. Класс `JoomproSubsControllerSubManager`

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↪reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

jimport('joomla.application.component.controlleradmin');
/**
 * Класс списка контроллеров для компонента Joomprosubs.
 *
 * @since 2.5
 */
class JoomproSubsControllerSubManager extends JControllerAdmin
{
    /**
     * Замещает получение модели.
     */
    public function getModel($name = 'Subscription', $prefix =
↪'JoomproSubsModel', $config = array('ignore_request' => true))
    {
```

```
        $model = parent::getModel($name, $prefix, $config);
        return $model;
    }
} // конец класса
```

Метод `getModel()` позволяет контроллеру получить подходящую модель (в данном случае из класса `JoomproSubsModelSubscription`). Он наследует методы `publish()` и `checkin()` из своего родительского класса `JControllerAdmin`. Поэтому они используются для выполнения упомянутых выше задач без всяких изменений и связанных с этим дополнительных затрат труда.

На панели инструментов имеются также пиктограммы для настройки параметров (Options) и получения справки (Help). Напомним, что при выборе пиктограммы Options из файла `config.xml` вызывается компонент `com_config`. Ниже приведено содержимое файла `config.xml`.

```
<?xml version="1.0" encoding="utf- 8"?>
<config>
    <fieldset name="permissions"
        description="JCONFIG_PERMISSIONS_DESC"
        label="JCONFIG_PERMISSIONS_LABEL"
    >
      <field name="rules" type="rules"
          component="COM_JOOMPROSUBS"
          filter="rules"
          validate="rules"
          label="JCONFIG_PERMISSIONS_LABEL"
          section="component" />
    </fieldset>
</config>
```

Если бы на уровне компонента имелись какие-нибудь параметры, они были бы указаны в этом файле и отображались бы на экране Options. Но в данном примере параметры у компонента отсутствуют, а имеются только полномочия, которые указываются в элементе разметки `field` типа `rules`.

Для создаваемого компонента необходимо также указать конкретные действия, создав файл `access.xml`. Ниже приведено содержимое этого файла.

```
<?xml version="1.0" encoding="utf-8"?>
<access component="com_joomprosubs">
    <section name="component">
      <action name="core.admin" title="JACTION_ADMIN"
↳description="JACTION_ADMIN_COMPONENT_DESC" />
      <action name="core.manage" title="JACTION_MANAGE"
↳description="JACTION_MANAGE_COMPONENT_DESC" />
      <action name="core.create" title="JACTION_CREATE"
↳description="JACTION_CREATE_COMPONENT_DESC" />
      <action name="core.delete" title="JACTION_DELETE"
↳description="JACTION_DELETE_COMPONENT_DESC" />
      <action name="core.edit" title="JACTION_EDIT"
↳description="JACTION_EDIT_COMPONENT_DESC" />
      <action name="core.edit.state" title="JACTION_EDITSTATE"
↳description="JACTION_EDITSTATE_COMPONENT_DESC" />
```

```
    <action name="core.edit.own" title="JACTION_EDITOWN"
↳description="JACTION_EDITOWN_COMPONENT_DESC" />
  </section>
  <section name="category">
    <action name="core.create" title="JACTION_CREATE"
↳description="COM_CATEGORIES_ACCESS_CREATE_DESC" />
    <action name="core.delete" title="JACTION_DELETE"
↳description="COM_CATEGORIES_ACCESS_DELETE_DESC" />
    <action name="core.edit" title="JACTION_EDIT"
↳description="COM_CATEGORIES_ACCESS_EDIT_DESC" />
    <action name="core.edit.state" title="JACTION_EDITSTATE"
↳description="COM_CATEGORIES_ACCESS_EDITSTATE_DESC" />
    <action name="core.edit.own" title="JACTION_EDITOWN"
↳description="COM_CATEGORIES_ACCESS_EDITOWN_DESC" />
  </section>
</access>
```

Своим содержимым этот файл почти ничем не отличается от аналогичного файла `access.xml` из компонента Weblinks, за исключением имени компонента во второй строке кода. Ведь в данном примере на уровнях компонента и категории выполняются те же самые действия, что и для компонента Weblinks. По завершении программирования рассматриваемого здесь компонента можно будет щелкнуть на пиктограмме Options, чтобы открыть для просмотра одноименный экран с полномочиями, приведенный на рис. 9.2.

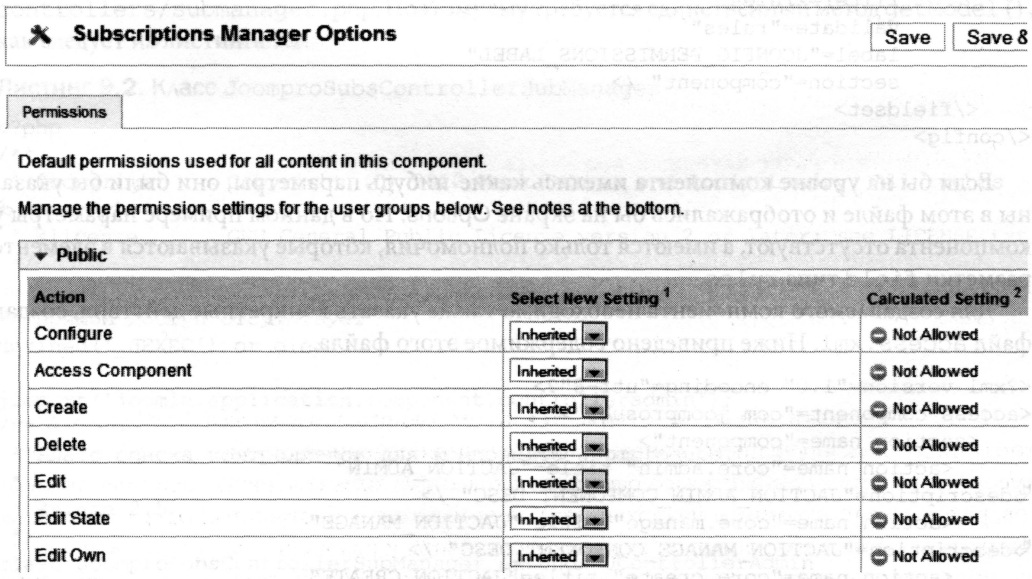


Рис. 9.2. Экран с полномочиями на подписку

Представление диспетчера

Представление для отображения экрана диспетчера называется `JoomproSubsViewSubManager` в соответствии с принятыми условными обозначениями, обсуждавшимися в главе 7. Третья часть имени этого представления (`SubManager` — Подчиненный

диспетчер) служит лишь для его описания. Это имя определено однозначно, поскольку в своей первой части оно содержит имя компонента JoomproSubs, отличающее его от любых других компонентов. Вместе с тем оно обозначает класс данного представления, находящийся в файле views/submanager/view.html.php, который также именуется в соответствии с условными обозначениями, принятыми для основных компонентов.

Ниже приведен исходный код из первой части объявления класса JoomproSubsViewSubManager в данном файле.

```
defined('_JEXEC') or die;
jimport('joomla.application.component.view');
/**
 *  Класс представления списка доступных подписок.
 *
 */
class JoomproSubsViewSubmanager extends JView
{
  protected $items;
  protected $pagination;
  protected $state;

  /**
   * Отображает представление.
   */
  public function display($tpl = null)
  {
    $this->state = $this->get('State');
    $this->items = $this->get('Items');
    $this->pagination = $this->get('Pagination');

    // проверить ошибки
    if (count($errors = $this->get('Errors'))) {
      JError::raiseError(500, implode("\n", $errors));
      return false;
    }

    $this->addToolbar();
    parent::display($tpl);
  }
```

И этот код почти ничем не отличается от исходного кода класса WeblinksViewWeblinks, за исключением, конечно, обозначения имени компонента JoomproSubs вместо Weblinks. Данный класс расширяет класс JView и содержит поля для элементов, разбиения на страницы и состояния. У него имеется единственный открытый метод display(). Этот метод получает элементы, состояние и разбиение на страницы из модели, проверяет ошибки, добавляет панель инструментов, а затем вызывает другой метод display() из своего родительского класса JView.

В данном классе определен еще один метод под названием addToolbar(). Ниже приведен исходный код этого метода.

```
/**
 * Метод, добавляющий заглавие страницы и панель инструментов.
 *
```

```
 * @since 2.5
 */
protected function addToolbar()
{
    JLoader::register('JoomproSubsHelper',
↳JPATH_COMPONENT.'/helpers/joomprosubs.php');
    $state = $this->get('State');
    $canDo = JoomprosubsHelper::getActions(
↳$state->get('filter.category_id'));
    $user = JFactory::getUser();

    JToolBarHelper::title(JText::_(
↳'COM_JOOMPROSUBS_MANAGER_JOOMPROSUBS'), 'newsfeeds.png');
    if (count($user->getAuthorisedCategories('com_joomprosubs',
↳'core.create')) > 0) {
      JToolBarHelper::addNew('subscription.add','JTOOLBAR_NEW');
    }
    if ($canDo->get('core.edit')) {
      JToolBarHelper::editList('subscription.edit','JTOOLBAR_EDIT');
    }

    if ($canDo->get('core.edit.state')) {

      JToolBarHelper::divider();
      JToolBarHelper::publish('submanager.publish',
↳'JTOOLBAR_PUBLISH', true);
      JToolBarHelper::unpublish('submanager.unpublish',
↳'JTOOLBAR_UNPUBLISH', true);

      JToolBarHelper::divider();
      JToolBarHelper::archiveList('submanager.archive');
      JToolBarHelper::checkin('submanager.checkin');
    }

    if ($state->get('filter.state') == -2 && $canDo->get('core.delete')) {
      JToolBarHelper::deleteList('', 'submanager.delete',
↳'JTOOLBAR_EMPTY_TRASH');
    JToolBarHelper::divider();
    } else if ($canDo->get('core.edit.state')) {
      JToolBarHelper::trash('submanager.trash','JTOOLBAR_TRASH');
      JToolBarHelper::divider();
    }

    if ($canDo->get('core.admin')) {
      JToolBarHelper::preferences('com_joomprosubs');
      JToolBarHelper::divider();
    }

    JToolBarHelper::help('', '', JText::_(
↳'COM_JOOMPROSUBS_SUBMANAGER_HELP_LINK'));
}
```

И этот код почти ничем не отличается от исходного кода соответствующего метода `addToolbar()` из компонента Weblinks. Сначала в данном методе загружается вспомогательный файл, используемый для создания объекта, в котором должны храниться действия, разрешенные текущему пользователю для выполнения. Затем отображается панель инструментов с пиктограммами разрешенных пользователю действий. Напомним, что проверок на уровне пользовательского интерфейса, а не самого метода оказывается достаточно для надежной защиты от попыток выполнить действие неуполномоченным пользователем. Эти проверки необходимо также повторить перед выполнением каждой задачи. Если пользователю разрешено выполнить действие `core.admin`, на панели инструментов добавляется пиктограмма Options. Напомним, что вместе с этой пиктограммой добавляется ссылка на компонент `com_config` с параметрами, указанными в файле `config.xml`.

И в последней строке кода из данного метода на панели инструментов добавляется пиктограмма Help. В этом случае указывается, что после щелчка кнопкой мыши на пиктограмме Help пользователь направляется по URL, указанному в файле языковой поддержки по ключу `COM_JOOMPROSUBS_SUBMANAGER_HELP_LINK`. Благодаря тому что значение, определяемое этим ключом, сделано зависимым от выбранного языка, можно без труда создать экран справки на другом языке. В методе `JToolBarHelper::help()` генерируется следующий код HTML для разметки страницы справки.

```
<a class="toolbar" rel="help" onclick="popupWindow(
'http://joomlaprogrammingbook.com/joompro-subscriptions-help.html', 'Help',
↳700, 500, 1)" href="#">
<span class="icon-32-help">
</span>
Help
</a>
```

Требующийся URL получается в результате перевода значения по языковому ключу `COM_JOOMPROSUBS_SUBMANAGER_HELP_LINK`. Если щелкнуть на пиктограмме Help, то по указанной ссылке во всплывающем окне откроется экран справки по данному компоненту, как показано на рис. 9.3.

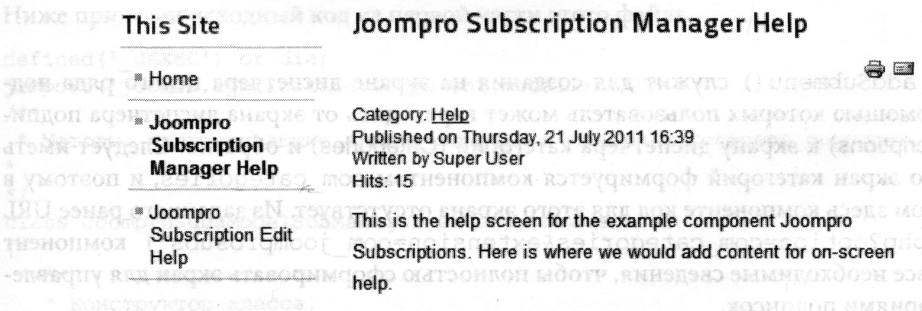


Рис. 9.3. Экран справки по компоненту подписки

По желанию пользователя можно также направить к локальному справочному файлу. Для этого следует указать имя справочного файла в качестве второго аргумента рассматриваемого здесь метода. И в этом случае для обращения к справочным файлам на разных языках можно воспользоваться методом `JText::_()`.

Вспомогательный класс

Следует заметить, что в предыдущем файле вызывался метод `Joomprosubs-Helper::getActions()` из файла `helpers/joomprosubs.php`. Ниже приведена первая часть исходного кода из этого вспомогательного файла.

```
defined('_JEXEC') or die;
/**
 * Вспомогательный класс Joomprosubs.
 *
 */
class JoomproSubsHelper
{
  /**
   *  Формирует панель ссылок.
   *
   * @param string           Имя активного представления.
 */
  public static function addSubmenu($vName = 'submanager')
  {
    JSubMenuHelper::addEntry(
        JText::_('COM_JOOMPROSUBS_SUBMENU_JOOMPROSUBS'),
        'index.php?option=com_joomprosubs&view=submanager',
        $vName == 'submanager'
    );
    JSubMenuHelper::addEntry(
        JText::_('COM_JOOMPROSUBS_SUBMENU_CATEGORIES'),
        'index.php?option=com_categories&extension=com_joomprosubs',
        $vName == 'categories'
    );
    if ($vName=='categories') {
        JToolBarHelper::title(
↳JText::sprintf('COM_CATEGORIES_CATEGORIES_TITLE',
↳JText::_('com_joomprosubs')),
        'joomprosubs-categories');
    }
  }
```

Метод `addSubmenu()` служит для создания на экране диспетчера целого ряда подменю, с помощью которых пользователь может переходить от экрана диспетчера подписок (Subscriptions) к экрану диспетчера категорий (Categories) и обратно. Следует иметь в виду, что экран категорий формируется компонентом `com_categories`, и поэтому в создаваемом здесь компоненте код для этого экрана отсутствует. Из заданного ранее URL (`'index.php?option=com_categories&extension=com_joomprosubs'`) компонент получает все необходимые сведения, чтобы полностью сформировать экран для управления категориями подписок.

Последним в данном вспомогательном классе определяется метод `getActions()`. Ниже приведен исходный код этого метода.

```
/**
 * Получает список выполняемых действий.
 *
```

```
 * @param   int         Идентификатор категории.
 * @return  JObject
 */
public static function getActions($categoryId = 0)
{
  $user = JFactory::getUser();
  $result = new JObject;

  if (empty($categoryId)) {
    $assetName = 'com_joomprosubs';
  } else {
    $assetName = 'com_joomprosubs.category.'.(int) $categoryId;
  }

  $actions = array(
   'core.admin', 'core.manage', 'core.create', 'core.edit',
   'core.edit.own', 'core.edit.state', 'core.delete'
  );

  foreach ($actions as $action) {
    $result->set($action, $user->authorise($action, $assetName));
  }

  return $result;
}
```

В этом методе создается объект, сообщающий, какие именно действия разрешено выполнять текущему пользователю. Получаемый в итоге результат используется в представлении для отображения конкретных панелей инструментов на экране диспетчера.

Модель диспетчера

В методе `display()` текущего представления из модели вызываются следующие три метода: `getState()`, `getItems()` и `getPagination()`. Модель для этого представления находится в классе `JoomproSubsModelSubManager` из файла `models/submanager.php`. Ниже приведен исходный код из первой части этого файла.

```
defined('_JEXEC') or die;
jimport('joomla.application.component.modellist');
/**
 * Методы, поддерживающие список записей в модели диспетчера подписок.
*
*/
class JoomproSubsModelSubManager extends JModelList
{
  /**
   * Конструктор класса.
   *
   * @param       array       Дополнительный ассоциативный массив
   *                          с параметрами настройки.
   * @see         JController
   * @since       2.5
   */
```

```
  public function __construct($config = array())
  {
    if (empty($config['filter_fields'])) {
      $config['filter_fields'] = array(
          'id', 'a.id',
          'title', 'a.title',
          'alias', 'a.alias',
          'checked_out', 'a.checked_out',
          'checked_out_time', 'a.checked_out_time',
          'catid', 'a.catid', 'category_title',
          'published', 'a. published ',
          'access', 'a.access', 'access_level',
          'created', 'a.created',
          'created_by', 'a.created_by',
          'publish_up', 'a.publish_up',
          'publish_down', 'a.publish_down',
          'group_title', 'g.title',
          'duration', 'a.duration'
      );
    }

    parent::__construct($config);
  }
```

Как и в компоненте `Weblinks`, класс этой модели расширяет класс `JModelList`. В его конструкторе сначала создается массив достоверных полей фильтрации, а затем вызывается конструктор родительского класса. Напомним, что массив `'filter_fields'` позволяет организовать защиту от умышленного внесения запросов SQL, отфильтровывая недостоверные имена столбцов в части запроса, определяющей упорядочение данных, извлекаемых из таблицы базы данных, с помощью оператора `ORDER BY`. Если злоумышленник попытается внедрить какой-нибудь код SQL в заполняемую форму или формируемый URL, этот злонамеренный код будет отсеян, поскольку он не совпадает с действительными значениями в данном массиве.

Далее в рассматриваемой здесь модели диспетчера определяется метод `populateState()`. Ниже приведен исходный код этого метода.

```
/**
 * Метод для автоматического заполнения состояния модели.
 *
 * Примечание: вызов метода getState() в этом методе приведет к рекурсии.
 *
 */
protected function populateState($ordering = null, $direction = null)
{
  // инициализировать переменные
  $app = JFactory::getApplication('administrator');

  // загрузить состояние фильтра
  $search = $this->getUserStateFromRequest($this->context
↳.'.filter.search', 'filter_search');
  $this->setState('filter.search', $search);
```

```
  $accessId = $this->getUserStateFromRequest($this->context
↳.'.filter.access', 'filter_access', null, 'int');
  $this->setState('filter.access', $accessId);

  $published = $this->getUserStateFromRequest($this->context
↳.'.filter.state', 'filter_published', '', 'string');
  $this->setState('filter.state', $published);

$categoryId = $this->getUserStateFromRequest($this->context
↳.'.filter.category_id', 'filter_category_id', '');
  $this->setState('filter.category_id', $categoryId);

  // загрузить параметры
  $params = JComponentHelper::getParams('com_joomprosubs');
  $this->setState('params', $params);

  // представить списком сведения о состоянии модели
  parent::populateState('a.title', 'asc');
}
```

Напомним, что в рассматриваемом здесь диспетчере имеются фильтры для поля поиска, состояния "опубликовано", категории и уровня доступа. В данном методе различные переменные состояния извлекаются из запроса и сохраняются в поле состояния модели. Кроме того, сохраняются параметры компонента. Затем из родительского класса вызывается метод `populateState()`, где на представленные списком сведения о состоянии накладываются ограничения и вводятся начальные ограничительные значения для листания списка. А для проверки достоверности упорядочиваемых данных используется массив `'filter_fields'`, созданный ранее в конструкторе класса данной модели.

Далее в рассматриваемой здесь модели диспетчера определяется метод `getStoreId()`. Ниже приведен исходный код этого метода.

```
/**
 * Метод для получения идентификатора сохранения, исходя из
 * состояния конфигурации модели.
 *
 * Это необходимо потому, что данная модель используется в компоненте
  * и различных модулях, где могут потребоваться разные наборы данных
  * или предъявлены разные требования к упорядочению.
 *
 * @param   string $id        Префикс идентификатора сохранения.
 * @return  string            Идентификатор сохранения.
 */
protected function getStoreId($id = '')
{
    // скомпилировать идентификатор сохранения
    $id.= ':' . $this->getState('filter.search');
    $id.= ':' . $this->getState('filter.access');
    $id.= ':' . $this->getState('filter.state');
    $id.= ':' . $this->getState('filter.category_id');

    return parent::getStoreId($id);
}
```

Напомним, что этот метод служит для создания однозначного ключа, предназначенного для кешируемого варианта списка элементов содержимого, объекта запроса, общего подсчета или разбиения на страницы. В данном случае однозначность списка элементов содержимого определяется из сочетания четырех доступных для использования фильтров и начала, ограничения, упорядочения и направления для упорядочения, причем четыре последних критерия вводятся в родительском методе `getStoreID()`, а полученный результат преобразуется в хеш-значение с помощью функции `md5()` и затем сохраняется. Если далее отображается точно такой же экран, как и сохраненный в оперативной памяти, то, используя его сохраненную копию, можно сэкономить немного времени на обработку. Ведь большая часть этой обработки выполняется в классе `JModelList`, а следовательно, остается лишь добавить в код метод `getStoreId()`, включающий все возможные значения фильтров.

И последним в рассматриваемой здесь модели диспетчера определяется метод `getListQuery()`. Ниже приведен исходный код этого метода.

```
/**
 * Формирует запрос SQL на загрузку данных списком.
 *
 * @return JDatabaseQuery
 */
protected function getListQuery()
{
  // создать новый объект запроса
  $db = $this->getDbo();
  $query = $db->getQuery(true);

  // выбрать требующиеся поля из таблицы
  $query->select('a.*');
  $query->from($db->quoteName('#__joompro_subscriptions').' AS a');

  // соединить для выбора среди пользователей того, кто снял
  // подписку с регистрации (заблокировал ее для редактирования)
  $query->select('uc.name AS editor');
  $query->join('LEFT', $db->quoteName('#__users').' AS uc
↳ON uc.id=a.checked_out');

  // соединить по группам пользователей, чтобы получить имя группы
  $query->select('g.title as group_title');
  $query->join('LEFT', $db->quoteName('#__usergroups').' AS g
↳ON a.group_id = g.id');

  // соединить по категориям
  $query->select('c.title AS category_title');
  $query->join('LEFT', $db->quoteName('#__categories').' AS c
↳ON c.id = a.catid');

  // отфильтровать по уровню доступа
  if ($access = $this->getState('filter.access')) {
    $query->where('a.access = '.(int) $access);
  }
```

```
    // отфильтровать по состоянию "опубликовано"
    $published = $this->getState('filter.state');
    if (is_numeric($published)) {
      $query->where('a.published = '.(int) $published);
    } else if ($published === '') {
      $query->where('(a.published IN (0, 1))');
    }

    // отфильтровать по категории
    $categoryId = $this->getState('filter.category_id');
    if (is_numeric($categoryId)) {
      $query->where('a.catid = '.(int) $categoryId);
    }

    // отфильтровать по критерию поиска в названии
    $search = $this->getState('filter.search');
    if (!empty($search)) {
      if (stripos($search, 'id:') === 0) {
        $query->where('a.id = '.(int) substr($search, 3));
      } else {
        $search = $db->Quote('%'.$db->getEscaped($search, true).'%');
        $query->where('(a.title LIKE '.$search.'
↳OR a.alias LIKE '.$search.')');
      }
    }

    // добавить оператор упорядочения списка
    $orderCol = $this->state->get('list.ordering');
    $orderDirn = $this->state->get('list.direction');
    $query->order($db->getEscaped($orderCol.' '.$orderDirn));

    return $query;
}
```

В данном методе формируется запрос на выборку строк из таблицы базы данных. С этой целью создается новый объект типа `JDatabaseQuery` при вызове метода `$db->getQuery(true)`. Следует особо отметить, что, начиная с версии Joomla 1.7, где используется платформа версии 11.1, класс `JDatabaseQuery` объявляется как абстрактный. Это означает, что получить экземпляр его объекта нельзя, а следовательно, и создать новый объект запроса, вызвав конструктор данного класса в операторе `new JDatabaseQuery()`. Но, вызывая метод `$db->getQuery(true)`, можно все же создать объект типа `JDatabaseQuery`, характерный для конкретного типа базы данных (например, `JDatabaseQueryMySQLi`). Именно такой объект и требуется в Joomla для работы с разнотипными базами данных.

Далее в рассматриваемом здесь методе выбираются все столбцы из таблицы (в данном случае — `#__joompro_subscriptions`) с помощью синтаксической конструкции `'a.*'` языка SQL. После этого осуществляется соединение с таблицей `#__users`, чтобы получить имя пользователя, снявшего подписку с регистрации для редактирования (т.е. заблокировавшего ее от правки другими пользователями). Соединяя таблицы `#__usergroups` и `#__categories`, можно отобразить названия группы пользователей и категории для подписок.

Особое внимание следует обратить на использование префикса #__ в запросах. Ведь заранее неизвестно, префикс какой именно таблицы будет использоваться на веб-сайте. Поэтому нужный префикс подставляется в Joomla перед тем, как отправлять запрос в базу данных. Следует также заметить, что имена таблиц указываются в качестве аргумента при вызове метода `$db->quoteName()`. Это дает возможность использовать разные символы кавычек для экранирования имен таблиц и столбцов при обращении к разнотипным базам данных. Например, в базе данных MySQL для этой цели используются обратные кавычки (`). Более подробно этот вопрос рассматривается в главе 11.

Все соединения являются левыми (LEFT), а это означает, что строка из главной таблицы, `#__joompro_subscriptions`, будет отображаться даже в том случае, если не найдено ни одной совпадающей строки ни в одной из упомянутых выше таблиц. Но в данном случае этого вообще не произойдет, поскольку категория, пользователь и группа пользователей должны быть всегда достоверными.

Далее проверяются уровень доступа, состояние "опубликовано", категория и фильтры поиска. Если все эти критерии поиска заданы, то в запрос базы данных вводится оператор WHERE с целью наложить определенные ограничения на результаты запроса. Следует заметить, что для поиска по идентификационному номеру имеется специальный код, в котором проверяется, совпадают ли три первых символа, введенных в поле поиска, с символьной строкой `"id:"`. Если они совпадают, то осуществляется поиск на совпадение с целочисленным идентификатором. В противном случае предполагается, что введенный текст частично совпадает с искомым названием. И наконец, в запрос вводится упорядочение получаемых в итоге данных в соответствии со значениями, определяющими само упорядочение и его направление в переменных состояния.

Таблицы базы данных

В предыдущем разделе был рассмотрен процесс формирования запроса к одной из новых таблиц базы данных. А теперь выясним, где же эти таблицы создаются. Для автоматического формирования таблиц при установке рассматриваемого здесь компонента создается файл `sql/install.mysql.utf8.sql`. Ниже приведено содержимое этого файла.

```
CREATE TABLE IF NOT EXISTS `#__joompro_subscriptions` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT 'Automatic
↳incrementing key field',
  `catid` int(11) NOT NULL DEFAULT '0' COMMENT 'Foreign key to
↳#__categories=table',
  `title` varchar(250) NOT NULL DEFAULT '' COMMENT 'Title of Subscription',
  `alias` varchar(255) NOT NULL DEFAULT '' COMMENT 'Alias value, used for
↳SEF URLs',
  `description` text NOT NULL COMMENT 'Description (will be edited using
↳editor)',
  `group_id` int(11) NOT NULL DEFAULT '0' COMMENT 'Foreign key to
↳#__usergroups table',
  `duration` int(11) NOT NULL DEFAULT '0' COMMENT 'Number for days that
↳subscription lasts',
  `published` tinyint(1) NOT NULL DEFAULT '0' COMMENT 'Published state
↳(1=published, 0=unpublished, - 2=trashed)',
  `checked_out` int(11) NOT NULL DEFAULT '0',
  `checked_out_time` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
```

```
  `access` int(11) NOT NULL DEFAULT '1' COMMENT 'Used to control access to
↳subscriptions',
  `params` text NOT NULL COMMENT 'For possible future use to add item-level
↳parameters (JSON string format)',
  `language` char(7) NOT NULL DEFAULT '' COMMENT 'For possible future use to
↳add language switching',
  `created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `created_by` int(10) unsigned NOT NULL DEFAULT '0' COMMENT 'Foreign key to
↳#__users table for user who created this item',
  `created_by_alias` varchar(255) NOT NULL DEFAULT '',
  `modified` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `modified_by` int(10) unsigned NOT NULL DEFAULT '0' COMMENT 'Foreign key
↳to #__users table for user who modified this item',
  `publish_up` datetime NOT NULL DEFAULT '0000-00-00 00:00:00' COMMENT 'Date
↳to start publishing this item',
  `publish_down` datetime NOT NULL DEFAULT '0000-00-00 00:00:00' COMMENT
↳'Date to stop publishing this item',
  PRIMARY KEY (`id`),
  KEY `idx_access` (`access`),
  KEY `idx_checkout` (`checked_out`),
  KEY `idx_published` (`published`),
  KEY `idx_catid` (`catid`),
  KEY `idx_createdby` (`created_by`),
  KEY `idx_language` (`language`)
  ) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

CREATE TABLE IF NOT EXISTS `#__joompro_sub_mapping` (
  `subscription_id` int(11) NOT NULL DEFAULT '0' COMMENT 'Foreign Key to
↳#_joompro_subscriptions.id',
  `user_id` int(11) NOT NULL DEFAULT '0' COMMENT 'Foreign Key to
↳#__users.id',
  `start_date` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `end_date` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`subscription_id`, `user_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

В этом файле создаются две новые таблицы базы данных. Обращение к нему будет организовано в XML-файле компоновки как к установочному файлу, автоматически выполняемому при установке данного компонента. В отношении этого файла необходимо отметить следующее.

- Это не исходный файл PHP. Он содержит сценарий базы данных SQL, т.е. набранную простым текстом последовательность команд языка SQL.
- Он специально предназначается для конкретной базы данных — MySQL. Для поддержки установки других баз данных будут предоставлены отдельные установочные файлы, аналогичные данному.
- Для обращения к таблицам в нем используется префикс `#__`. Поэтому он специально предназначен для выполнения в среде Joomla, где объект типа `JDatabase` преобразует этот префикс в подходящий для каждого веб-сайта на Joomla. (В целях проверки можете выполнить сценарий из этого файла относительно тестовой базы данных, используя такую программу управления базой данных, как, например,

phpMyAdmin. Для этого достаточно загрузить данный файл в текстовый редактор и выполнить операцию поиска и замены префикса #__ соответствующим префиксом таблиц базы данных для вашего веб-сайта на Joomla.)

Более подробно вопросы создания таблиц базы данных будут рассматриваться в главе 11. А до тех пор вы можете создать одну из таких таблиц, установив рассматриваемый здесь компонент из загруженного архивного файла или выполнив приведенный выше сценарий базы данных SQL из программы phpMyAdmin, но предварительно заменив префикс в установочном файле, как пояснялось выше.

Если пользователь удаляет данный компонент, то сохранять таблицы в базе данных нецелесообразно. Поэтому для этой цели предоставляется отдельный файл `sql/uninstall.mysql.utf8.sql`, содержащий следующие строки кода:

```
DROP TABLE IF EXISTS '#__joompro_subscriptions';
DROP TABLE IF EXISTS '#__joompro_sub_mapping';
```

В этих строках кода из базы данных удаляются две таблицы. Более подробно они рассматриваются в главе 11.

Компоновка экрана диспетчера

И последняя программа для отображения экрана диспетчера выполняет его компоновку из файла `views/submanager/tmpl/default.php`. Как и все остальные файлы, этот файл составлен по образцу соответствующего файла компоновки для компонента Weblinks (`administrator/components/com_weblinks/views/weblinks/tmpl/default.php`). Ниже приведена первая часть кода из этого файла компоновки.

```
defined('_JEXEC') or die;
JHtml::addIncludePath(JPATH_COMPONENT.'/helpers/html');
JHtml::_('behavior.tooltip');
JHtml::_('script','system/multiselect.js', false, true);

$user = JFactory::getUser();
$userId = $user->get('id');
$listOrder = $this->escape($this->state->get('list.ordering'));
$listDirn = $this->escape($this->state->get('list.direction'));
?>
```

В этой части файла компоновки выполняются все необходимые организационные действия и получается текущий способ упорядочения списка. Ниже приведена следующая часть кода из данного файла компоновки.

```
<div class="joomprosubs-manager">
<form action="<?php echo
↳JRoute::_('index.php?option=com_joomprosubs&view=submanager'); ?>"
  method="post" name="adminForm" id="adminForm">
  <fieldset id="filter-bar">
    <div class="filter-search fltlft">
      <label class="filter-search-lbl" for="filter_search"><?php echo
↳JText::_('JSEARCH_FILTER_LABEL'); ?></label>
      <input type="text" name="filter_search" id="filter_search"
↳value="<?php echo $this->escape($this->state->get('filter.search')); ?>"
```

```
title="<?php echo JText::_('COM_JOOMPROSUBS_SEARCH_IN_TITLE'); ?>" />
      <button type="submit"><?php echo JText::_('JSEARCH_FILTER_SUBMIT');
?></button>
      <button type="button" onclick=
"document.id('filter_search').value='';this.form.submit();">
<?php echo JText::_('JSEARCH_FILTER_CLEAR'); ?></button>
      </div>
      <div class="filter-select fltrt">

        <select name="filter_published" class="inputbox"
onchange="this.form.submit()">
          <option value=""><?php echo
JText::_('JOPTION_SELECT_PUBLISHED');?></option>
          <?php echo JHtml::_('select.options',
JHtml::_('jgrid.publishedOptions'), 'value', 'text',
$this->state->get('filter.state'), true);?>
        </select>

        <select name="filter_category_id" class="inputbox"
onchange="this.form.submit()">
<option value=""><?php echo
JText::_('JOPTION_SELECT_CATEGORY');?></option>
          <?php echo JHtml::_('select.options',
JHtml::_('category.options', 'com_joomprosubs'), 'value', 'text',
$this->state->get('filter.category_id'));?>
        </select>

            <select name="filter_access" class="inputbox"
onchange="this.form.submit()">
          <option value=""><?php echo
JText::_('JOPTION_SELECT_ACCESS');?></option>
          <?php echo JHtml::_('select.options',
JHtml::_('access.assetgroups'), 'value', 'text',
$this->state->get('filter.access'));?>
        </select>

      </div>
  </fieldset>
<div class="clr"> </div>
```

В этой части исходного кода компонуются средства поиска и другие фильтры. Он написан по образцу исходного кода из файла компоновки для компонента Weblinks. Обратите внимание на то, что весь класс компоновки под именем `joomprosubs-manager` заключен внутри элемента разметки `div`. Это дает разработчикам возможность выполнить специальное оформление экрана диспетчера с помощью вложенных таблиц стилей (CSS).

Ниже приведена следующая часть кода из данного файла компоновки.

```
<table class="adminlist">
  <thead>
    <tr>
      <th style="width: 1%;">
        <input type="checkbox" name="checkall-toggle" value=""
onclick="checkAll(this)" />
```

```
      </th>
      <th class="title">
        <?php echo JHtml::_('grid.sort', 'JGLOBAL_TITLE', 'a.title',
$listDirn, $listOrder); ?>
      </th>
      <th style="width: 5%;">
        <?php echo JHtml::_('grid.sort', 'JSTATUS', 'a.published',
$listDirn, $listOrder); ?>
      </th>

      <th style="width: 20%;">
        <?php echo JHtml::_('grid.sort', 'JCATEGORY',
'category_title', $listDirn, $listOrder); ?>
      </th>

      <th style="width: 20%;">
                        <?php echo JHtml::_('grid.sort',
'COM_JOOMPROSUBS_FIELD_USERGROUP_LABEL', 'g.title', $listDirn,
$listOrder); ?>
      </th>

      <th style="width: 10%;">
        <?php echo JHtml::_('grid.sort',
'COM_JOOMPROSUBS_FIELD_DURATION_LABEL', 'a.duration', $listDirn,
$listOrder); ?>
      </th>

      <th style="width: 5%;">
        <?php echo JHtml::_('grid.sort', 'JGRID_HEADING_ACCESS',
'a.access', $listDirn, $listOrder); ?>
      </th>

      <th style="width: 5%;" class="nowrap">
        <?php echo JHtml::_('grid.sort', 'JGRID_HEADING_ID', 'a.id',
$listDirn, $listOrder); ?>
      </th>
    </tr>
  </thead>
```

В этой части исходного кода компонуются сортируемые заголовки столбцов подобно тому, как это делается на экране Weblinks Manager. Но в отличие от его компоновки, в данном случае применяется более предпочтительный синтаксис встраиваемого стилевого оформления столбцов по ширине.

В следующей части данного файла компонуется разбиение на страницы у нижнего края экрана. Ниже приведен исходный код этой части файла.

```
<tfoot>
  <tr>
    <td colspan="10">
      <?php echo $this->pagination->getListFooter(); ?>
    </td>
  </tr>
</tfoot>
```

Следующая часть данного файла компоновки начинается с цикла `foreach`, в котором отдельные подписки обрабатываются по списку, как показано ниже.

```
<tbody>
<?php foreach ($this->items as $i => $item) :
  $ordering = ($listOrder == 'a.ordering');
  $item->cat_link =
↳JRoute::_('index.php?option=com_categories&extension=com_joomprosubs
↳&task=edit&type=other&cid[]='. $item->catid);
  $canCreate = $user->authorise('core.create',
↳'com_joomprosubs.category.'.$item->catid);
  $canEdit = $user->authorise('core.edit',
↳'com_joomprosubs.category.'.$item->catid);
  $canCheckin = $user->authorise('core.manage', 'com_checkin') ||
↳$item->checked_out==$user->get('id') || $item->checked_out==0;
  $canChange = $user->authorise('core.edit.state',
↳'com_joomprosubs.category.'.$item->catid) && $canCheckin;
  ?>
```

В этой части исходного кода компоновки проверяется, разрешено ли текущему пользователю создавать, редактировать, регистрировать и публиковать каждый элемент списка. А в следующей части исходного кода из данного файла компоновки отображаются сведения о каждой подписке из списка, как показано ниже.

```
    <tr class="row<?php echo $i % 2; ?>">
      <td class="center">
        <?php echo JHtml::_('grid.id', $i, $item->id); ?>
      </td>
      <td>
        <?php if ($item->checked_out) : ?>
        <?php echo JHtml::_('jgrid.checkedout', $i,
↳$item->editor, $item->checked_out_time, 'submanager.', $canCheckin); ?>
        <?php endif; ?>
        <?php if ($canEdit) : ?>
          <a href="<?php echo
↳JRoute::_('index.php?option=com_joomprosubs&task=subscription.edit&id='.
↳(int) $item->id); ?>">
          <?php echo $this->escape($item->title); ?></a>
        <?php else : ?>
          <?php echo $this->escape($item->title); ?>
        <?php endif; ?>
        <p class="smallsub">
        <?php echo JText::sprintf('JGLOBAL_LIST_ALIAS',
↳$this->escape($item->alias));?></p>
      </td>
      <td class="center">
        <?php echo JHtml::_('jgrid.published', $item->published,
↳$i, 'submanager.', $canChange, 'cb', $item->publish_up,
↳$item->publish_down); ?>
      </td>

      <td class="center">
        <?php echo $this->escape($item->category_title); ?>
```

```
      </td>
      <td class="center">
        <?php echo $this->escape($item->group_title); ?>
      </td>

      <td class="center">
        <?php echo $this->escape($item->duration); ?>
      </td>

      <td class="center">
        <?php echo $this->escape($item->access); ?>
      </td>

      <td class="center">
        <?php echo (int) $item->id; ?>
      </td>
    </tr>
    <?php endforeach; ?>
  </tbody>
</table>
```

В этой части исходного кода компоновки каждая строка таблицы извлекается из объекта `$item` и отображается в ячейке таблицы, формируемой внутри элемента `td` разметки в коде HTML. И в последней части исходного кода из рассматриваемого здесь файла компоновки на экран выводятся скрытые поля задачи (`task`), установки флажка (`boxchecked`), упорядочения фильтров (`filter_order`) и направления для упорядочения фильтров (`filter_order_dir`), как показано ниже.

```
  <div>
    <input type="hidden" name="task" value="" />
    <input type="hidden" name="boxchecked" value="0" />
    <input type="hidden" name="filter_order" value="<?php echo
↳$listOrder; ?>" />
    <input type="hidden" name="filter_order_Dir" value="<?php echo
↳$listDirn; ?>" />
    <?php echo JHtml::_('form.token'); ?>
  </div>
</form>
</div>
```

Эта часть исходного кода завершается закрытием исходного элемента `div` разметки возможного стилевого оформления. Как видите, данный файл компоновки содержит немало кода, но этот код написан точно по образцу аналогичных файлов компоновки для компонента Weblinks и других базовых компонентов. Итак, у нас уже имеются все файлы, необходимые для выполнения выбираемой по умолчанию задачи, которая заключается в отображении экрана диспетчера.

Экран Subscriptions Manager: Add and Edit

А теперь рассмотрим код для ввода и редактирования подписки. Для этой цели используются следующие файлы, обсуждаемые в данном разделе.

- Файл controllers/subscription.php (класс JoomprosubsControllerSubscription).
- Файл views/subscription/view.html.php (класс JoomprosubsViewSubscription).
- Файл models/subscription.php (класс JoomproSubsModelSubscription).
- Файл views/subscription/tmpl/edit.php.
- Файл models/forms/subscription.xml.
- Файл tables/subscription.php (класс JoomprosubsTableSubscription).

Задачи контроллера

Для ввода и редактирования подписки из панели инструментов на экране диспетчера выбираются задачи subscription.add и subscription.edit соответственно. Как пояснялось в главе 7, это означает, что для выполнения первой задачи вызывается метод add(), а для выполнения второй задачи — метод edit() из класса контроллера (в данном случае из класса JoomprosubsControllerSubscription, объявляемого в файле controllers/subscription.php). Класс этого контроллера расширяет класс JControllerForm и лишь переопределяет два метода allowAdd() и allowEdit(), которые вызываются из методов add() и edit() класса JControllerForm для проверки полномочий пользователя на выполнение действий по вводу и редактированию подписки.

Ниже приведена первая часть исходного кода из файла controllers/subscription.php, где метод allowAdd() переопределяется в классе JoomprosubsControllerSubscription.

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

jimport('joomla.application.component.controllerform');
/**
 * Класс контроллера для компонента Joomprosubs.
 *
 */
class JoomproSubsControllerSubscription extends JControllerForm
{
  /**
   * Переменная списка представлений URL.
   *
   * @var   string
  */
  protected $view_list = 'submanager';

  /**
   * Переопределяемый метод для проверки полномочий
   * пользователя на ввод новой записи.
   *
   * @param       array $data       Массив исходных данных.
   * @return            boolean
   */
  protected function allowAdd($data = array())
  {
```

```
    // инициализировать переменные
    $user = JFactory::getUser();
$categoryId = JArrayHelper::getValue($data, 'catid',
$JRequest::getInt('filter_category_id'), 'int');
    $allow = null;

    if ($categoryId) {
      // проверить категорию, если она передана в URL
      $allow = $user->authorise('core.create',
$$this->option.'.category.'.$categoryId);
    }

    if ($allow === null) {
      // вернуться к полномочиям на уровне компонента
      // в отсутствие лучшей информации
      return parent::allowAdd($data);
    } else {
      return $allow;
    }
}
```

В приведенном выше коде определяется сам класс и первый его метод. Следует также заметить, что в этом классе определяется поле `$view_list`, в котором устанавливается значение `"submanager"`. В классе `WeblinksControllerWeblink` это поле не требовалось, поскольку для исходного представления по умолчанию выбиралось имя компонента в единственном числе (`Weblink`). Но в данном случае это правило не действует. Ведь если бы исходное представление называлось `joomprosubs`, поле `$view_list` или `$default_view`, определяемое в классе `JoomproSubsController`, не потребовалось бы. Но две лишние строки кода — совсем небольшая плата за получение осмысленных имен класса и файла, а кроме того, имя `submanager` имеет больший смысл, чем имя `joomprosubs`.

Метод `allowAdd()` очень похож на аналогичный метод из класса `WeblinksControllerWeblink`. В нем предпринимается попытка извлечь идентификатор категории из запроса. Он может быть получен в результате фильтрации исходных данных по категории. Если идентификатор категории обнаружен, то далее проверяется, разрешено ли текущему пользователю вводить элемент в этой категории. Если это ему не разрешено, то проверяются полномочия на уровне компонента, для чего вызывается метод `allowAdd()` из родительского класса.

В остальной части исходного кода рассматриваемого здесь класса переопределяется метод `allowEdit()`, как показано ниже.

```
  /**
   * Переопределяемый метод для проверки полномочий пользователя
   * на редактирование записи.
   *
   * @param       array $data       Массив исходных данных.
   * @param       string      $key              Имя первичного ключа.
   *
   * @return boolean
   */
  protected function allowEdit($data = array(), $key = 'id')
  {
```

```
    // инициализировать переменные
    $recordId = (int) isset($data[$key]) ? $data[$key] : 0;
    $categoryId = 0;

    if ($recordId) {
      $categoryId = (int) $this->getModel()->getItem($recordId)->catid;
    }

    if ($categoryId) {
      // если категория задана, проверить полномочия на нее
      return JFactory::getUser()->authorise('core.edit',
↳$this->option.'.category.'.$categoryId);
    } else {
      // вернуться к полномочиям на уровне компонента, поскольку
      // ресурсы не отслеживаются
      return parent::allowEdit($data, $key);
    }
  }
}
```

Как правило, при выполнении этого метода уже должен быть в наличии достоверный идентификатор подписки. В данном случае идентификатор категории извлекается из подписки и затем проверяются полномочия текущего пользователя на редактирование подписок данной категории. Если же достоверный идентификатор подписки отсутствует по какой-нибудь причине, то проверяются полномочия данного пользователя на редактирование подписок на уровне компонента.

Представление ввода и редактирования подписки

Если пользователю разрешено вводить и редактировать подписки, то в родительском контроллере устанавливается переадресация по следующему URL:

```
administrator/index.php?option=com_joomprosubs&view=subscription&layout=edit
```

Это приведет к выполнению задачи отображения средствами класса `Joompro SubsViewSubscription` из файла `views/subscription/view.html.php`. Ниже приведен код из первой части этого файла.

```
defined('_JEXEC') or die;
jimport('joomla.application.component.view');

/**
 * Представление для редактирования подписки.
 *
 * @package       Joomla.Administrator
 * @subpackage    com_joomprosubs
 */
class JoomprosubsViewSubscription extends JView
{
  protected $form;
  protected $item;
  protected $state;
  /**
```

```
   * Отображает представление
  */
  public function display($tpl = null)
  {
    // инициализировать переменные
    $this->form = $this->get('Form');
    $this->item = $this->get('Item');
    $this->state = $this->get('State');

    // проверить ошибки
    if (count($errors = $this->get('Errors'))) {
      JError::raiseError(500, implode("\n", $errors));
      return false;
    }

    $this->addToolbar();
    parent::display($tpl);
}
```

Эта часть упомянутого выше файла аналогична содержимому соответствующего файла для компонента Weblinks. Здесь сначала объявляется класс, расширяющий класс `JView`, а затем определяется метод `display()`. В этом методе сначала получается объект формы, элемент и его состояние из модели, а затем проверяются ошибки. Если ошибки не обнаружены, экранная форма дополняется панелью инструментов и затем из родительского класса вызывается соответствующий метод отображения.

Ниже приведен исходный код из последней части файла `views/subscription/view.html.php`.

```
protected function addToolbar()
{
  JRequest::setVar('hidemainmenu', true);

  $user = JFactory::getUser();
  $isNew = ($this->item->id == 0);
  $checkedOut = !($this->item->checked_out == 0 ||
↳$this->item->checked_out == $user->get('id'));
  $canDo = JoomprosubsHelper::getActions(
K$this->state->get('filter.category_id'), $this->item->id);

  JToolBarHelper::title(JText::_('COM_JOOMPROSUBS_MANAGER_JOOMPROSUB'),
↳'newfeeds.png');

  // проверить, можно ли сохранить элемент, если он не снят с регистрации
  if (!$checkedOut && ($canDo->get('core.edit')||
↳ (count($user->getAuthorisedCategories('com_joomprosubs',
↳'core.create')))))
  {
    JToolBarHelper::apply('subscription.apply', 'JTOOLBAR_APPLY');
    JToolBarHelper::save('subscription.save', 'JTOOLBAR_SAVE');
  }
  if (!$checkedOut && (count($user->getAuthorisedCategories(
↳'com_joomprosubs', 'core.create')))){
    JToolBarHelper::custom('subscription.save2new', 'save-new.png',
```

```
↳'save-new_f2.png', 'JTOOLBAR_SAVE_AND_NEW', false);
  }
  // проверить, можно ли сохранить копию, если элемент уже существует
  if (!$isNew && (count($user->getAuthorisedCategories(
↳'com_joomprosubs', 'core.create')) > 0)) {
    JToolBarHelper::custom('subscription.save2copy', 'save-copy.png',
↳'save-copy_f2.png', 'JTOOLBAR_SAVE_AS_COPY', false);
  }
  if (empty($this->item->id)) {
    JToolBarHelper::cancel('subscription.cancel', 'JTOOLBAR_CANCEL');
  }
  else {
    JToolBarHelper::cancel('subscription.cancel', 'JTOOLBAR_CLOSE');
  }

  JToolBarHelper::divider();
  JToolBarHelper::help('', '',
↳JText::_('COM_JOOMPROSUBS_SUBSCRIPTION_HELP_LINK'));
  }
}
```

Как и в соответствующем методе для компонента Weblinks, в приведенном выше методе `addToolbar()` проверяются полномочия пользователя, прежде чем отображать пиктограммы Save, Save & Close, Save & New и Save as Copy на панели инструментов. Здесь же отображаются пиктограммы Cancel/Close и Help. Напомним, что это делается лишь для усовершенствования пользовательского интерфейса, но не подменяет собой повторную проверку полномочий текущего пользователя на выполнение действий, обозначаемых упомянутыми пиктограммами, после того, как они будут начаты.

Модель ввода и редактирования подписок

В рассматриваемом здесь представлении из модели вызываются методы `getForm()`, `getItem()` и `getState()`, поэтому рассмотрим далее класс модели `JoomproSubsModelSubscription`, объявляемый в файле `models/subscription.php`. Напомним, что методы `getForm()` и `getItem()` наследуются в этом классе модели из классов `JModelAdmin` и `JModel` соответственно. Кроме того, в нем определяются еще шесть методов.

В первой части файла `models/subscription.php` определяется сам класс модели, а также метод `canDelete()`, как показано ниже.

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

jimport('joomla.application.component.modeladmin');

/**
 * Класс модели для компонента Joomprosubs.
 *
 * @package       Joomla.Administrator
 * @subpackage    com_joomprosubs
 * @since         2.5
*/
class JoomproSubsModelSubscription extends JModelAdmin
```

```
{
  /**
   * @var   string      Префикс, используемый в сообщениях контроллера
   */
  protected $text_prefix = 'COM_JOOMPROSUBS';

  /**
   * Метод для проверки полномочий на удаление записи.
   *
   * @param       object            Объект записи.
   * @return            boolean     Истинно, если запись разрешается удалить.
   *                                По умолчанию соответствует полномочиям,
   *                                установленным на уровне компонента.
   */
  protected function canDelete($record)
  {
    if (!empty($record->id)) {
      if ($record->published != -2) {
        return ;
      }
      $user = JFactory::getUser();

      if ($record->catid) {
        return $user->authorise('core.delete',
↳'com_joomprosubs.category.'.(int) $record->catid);
      }
      else {
        return parent::canDelete($record);
      }
    }
  }
```

Метод `canDelete()` вызывается при выполнении задачи. В нем просто проверяется, разрешено ли пользователю удалить элемент. Однако эта проверка полномочий производится лишь на уровне категории. В Joomla система списков контроля доступа (ACL) имеет иерархический характер, и поэтому полномочия достаточно проверить лишь на уровне категории. Так, если на уровне категории не установлены полномочия на выполнение данного действия, автоматически будут унаследованы полномочия, заданные на уровне компонента. Это избавляет от необходимости проверять полномочия на обоих уровнях иерархии доступа.

Далее в рассматриваемом здесь классе модели определяется метод `canEditState()`, как показано ниже.

```
/**
 * Метод для проверки возможного изменения состояния записи.
 *
 * @param   object      Объект записи.
 * @return  boolean     Истинно, если разрешается изменить
 *                      состояние записи. По умолчанию соответствует
 *                      полномочиям, установленным на уровне компонента.
 */
protected function canEditState($record)
```

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

  if (!empty($record->catid)) {
    return $user->authorise('core.edit.state',
'com_joomprosubs.category.'.(int) $record->catid);
  }
  else {
    return parent::canEditState($record);
  }
}
```

Этот метод вызывается всякий раз, когда изменяется состояние "опубликовано" редактируемого элемента. И в этом методе только проверяется, разрешено ли пользователю изменить состояние данного элемента.

Ниже приведен исходный код следующего далее метода `getTable()`.

```
/**
 * Возвращает ссылку на объект типа Table и всегда создает его.
 *
 * @param   type        Тип таблицы для получения экземпляра ее объекта.
 * @param   string      Необязательный префикс для имени табличного класса.
 * @param   array       Необязательный массив конфигурации модели.
 * @return  JTable      Объект базы данных.
 */
public function getTable($type = 'subscription',
$prefix = 'JoomproSubsTable', $config = array())
{
  return JTable::getInstance($type, $prefix, $config);
}
```

Этот метод вызывается всякий раз, когда предполагается запись данных в таблицу. Он всего лишь предоставляет нужное имя таблицы и получает экземпляр ее объекта из класса `JTable`.

Далее следует приведенный ниже метод `getForm()`.

```
/**
 * Метод для получения формы записи.
 *
 * @param   array $data       Необязательный массив данных
 *                                    для опроса в форме.
 * @param   boolean     loadData          Истинно, если в форму должны быть
 *                                        загружены ее собственные данные
 *                                        (по умолчанию), иначе – ложно.
 * @return  JForm                   Объект типа JForm, возвращаемый при
 *                                  успешном завершении операции,
 *                                  иначе – логическое значение false.
*/
public function getForm($data = array(), $loadData = true)
{
  // инициализировать переменные
  $app = JFactory::getApplication();

  // получить форму
```

```
  $form = $this->loadForm('com_joomprosubs.subscription',
↳'subscription', array('control' => 'jform', 'load_data' => $loadData));
  if (empty($form)) {
    return false;
  }

  // определить правильность проверяемых полномочий
  if ($this->getState('subscription.id')) {
    // Существующая запись. Может быть отредактирована
    // только в выбранных категориях.
    $form->setFieldAttribute('catid', 'action', 'core.edit');
  } else {
    // Новая запись. Может быть создана только в выбранных категориях.
    $form->setFieldAttribute('catid', 'action', 'core.create');
  }

  // видоизменить форму на основании контроля прав доступа
  if (!$this->canEditState((object) $data)) {
    // запретить отображение полей
    $form->setFieldAttribute('published', 'disabled', 'true');
    $form->setFieldAttribute('publish_up', 'disabled', 'true');
   $form->setFieldAttribute('publish_down', 'disabled', 'true');

    // сделать поля недоступными на время сохранения.
    // В контроллере уже проверено, что эту запись можно редактировать.
    $form->setFieldAttribute('published', 'filter', 'unset');
    $form->setFieldAttribute('publish_up', 'filter', 'unset');
    $form->setFieldAttribute('publish_down', 'filter', 'unset');
  }

  return $form;
}
```

Этот метод вызывается в двух местах цикла.

- При открытии формы для редактирования.
- При сохранении данных формы.

В этом методе сначала получаются данные для формы и проверяются полномочия пользователя на редактирование или создание элемента. Затем в нем проверяется, разрешено ли пользователю редактировать состояние подписки. Если у него нет таких полномочий, то выполняются следующие действия.

- Поля формы делаются недоступными и не отображаются в форме, для чего в атрибуте `disabled` устанавливается логическое значение `true`.
- Любые изменения, которые могут быть внесены в этих полях, отфильтровываются, для чего в атрибуте `filter` устанавливается значение `unset`.



Как пояснялось ранее, вводимые пользователем данные следует непременно проверить на достоверность, прежде чем приступать к их обработке. Даже если форма и не допускает ввод пользователем определенной информации, злоумышленник может без особого труда видоизменить разметку веб-страницы в коде HTML, чтобы внести из-

менения в форму и обойти данную проверку. Именно поэтому необходимо также проверять введенные в форме значения после ее передачи на обработку.

Класс `JForm` позволяет очень просто организовать такую проверку. Установкой значения `"unset"` в атрибуте `filter` классу `JForm` предписывается удалить любые данные, которые могли быть введены пользователем. В этом случае можно выяснить, что эти данные могли быть введены в браузере кем-то, видоизменившим форму после того, как поля формы были сделаны недоступными для ввода в обычном режиме. Эта надежная мера защиты от подобного рода злонамеренных попыток реализуется в коде данного класса, не требуя дополнительной проверки. Она служит характерным примером применения класса `JForm` для динамического изменения состояния и защиты от взлома.

Далее следует приведенный ниже метод `getFormData()`.

```
/**
 * Метод для получения данных, которые должны быть внесены в форму.
 *
 * @return  mixed       Данные для формы.
 */
protected function loadFormData()
{
  // проверить наличие введенных ранее данных в текущем сеансе работы
  $data = JFactory::getApplication()->getUserState
('com_joomprosubs.edit.subscription.data', array());

  if (empty($data)) {
    $data = $this->getItem();

    // заполнить форму рядом выбираемых по умолчанию значений
    if ($this->getState('subscription.id') == 0) {
        $app = JFactory::getApplication();
        $data->set('catid', JRequest::getInt('catid',
$app->getUserState('com_joomprosubs.submanager.filter.category_id')));
    }
  }

  return $data;
}
```

Сначала в этом методе получаются любые данные, которые могли быть сохранены в текущем сеансе работы. Напомним, что это делается для того, чтобы пользователю не пришлось еще раз заполнять форму полностью, если данные в каком-нибудь из ее полей окажутся недействительными. Если же в текущем сеансе работы не окажется введенных данных, они получаются с помощью метода `getItem()`. А если вводится новый элемент содержимого, то проверяется потребность его фильтрации по категории, и в таком случае идентификатор категории выбирается для формы по умолчанию.

И в последней части рассматриваемого здесь класса определяется метод `prepareTable()`, как показано ниже.

```
  /**
   * Подготавливает и подвергает таблицу дополнительной
   * обработке перед сохранением.
   *
```

```
   */
  protected function prepareTable(&$table)
  {
    $table->alias = JApplication::stringURLSafe($table->alias);
    if (empty($table->alias)) {
      $table->alias = JApplication::stringURLSafe($table->title);
    }
  }
} // конец класса
```

В этом методе проверяется, насколько безопасным для URL может оказаться столбец `alias`, содержащий псевдоним в подготавливаемой таблице. Ведь псевдоним используется в URL, если установлена кнопка-переключатель Yes параметра Search Engine Friendly URLs на вкладке SEO диалогового окна Global Configuration. Кроме того, URL автоматически устанавливается по названию подписки, если псевдоним не задан. В этом случае пользователь может пропустить поле ввода псевдонима, если желает употреблять название в качестве псевдонима.

Форма ввода и редактирования подписок

При воспроизведении формы на экране ее компоновка выбирается из файла `views/subscription/tmpl/edit.php`. Этот файл содержит сценарий, а не объявление класса. Ниже приведена первая часть исходного кода из этого файла компоновки.

```
<?php
/**
 * @copyright     Copyright (C) 2011 Mark Dexter and Louis Landry. All rights
↳reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.
txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

JHtml::addIncludePath(JPATH_COMPONENT.'/helpers/html');
JHtml::_('behavior.tooltip');
JHtml::_('behavior.formvalidation');
?>
<script type="text/javascript">
  Joomla.submitbutton = function(task)
  {
    if (task == 'subscription.cancel' ||
↳document.formvalidator.isValid(document.id('subscription-form'))) {
      <?php echo $this->form->getField('description')->save(); ?>
      Joomla.submitform(task, document.getElementById(
↳'subscription-form'));
    }
    else {
      alert('<?php echo $this->escape(
↳JText::_('JGLOBAL_VALIDATION_FORM_FAILED'));?>');
    }
  }
</script>
```

Как и в предыдущих примерах применения класса JForm, в данную компоновку добавляются два вида поведения: всплывающая подсказка и проверка достоверности формы. Затем определяется сценарий JavaScript, в котором будет производиться проверка достоверности формы при ее передаче на обработку. Более подробно эти вопросы рассматриваются в главе 12.

Ниже приведена следующая часть исходного кода из рассматриваемого здесь файла компоновки формы ввода и редактирования подписок.

```
<form action="<?php echo
JRoute::_('index.php?option=com_joomprosubs&layout=edit&id='.(int)
$this->item->id); ?>" method="post" name="adminForm"
id="subscription-form" class="form-validate">
<div class="width-60 fltlft">
  <fieldset class="adminform">
    <legend><?php echo empty($this->item->id) ?
JText::_('COM_JOOMPROSUBS_NEW_JOOMPROSUB') :
JText::sprintf('COM_JOOMPROSUBS_EDIT_JOOMPROSUB', $this->item->id); ?>
</legend>
    <ul class="adminformlist">
    <li><?php echo $this->form->getLabel('title'); ?>
    <?php echo $this->form->getInput('title'); ?></li>

    <li><?php echo $this->form->getLabel('alias'); ?>
    <?php echo $this->form->getInput('alias'); ?></li>

    <li><?php echo $this->form->getLabel('catid'); ?>
    <?php echo $this->form->getInput('catid'); ?></li>

    <li><?php echo $this->form->getLabel('group_id'); ?>
    <?php echo $this->form->getInput('group_id'); ?></li>

    <li><?php echo $this->form->getLabel('duration'); ?>
    <?php echo $this->form->getInput('duration'); ?></li>

    <li><?php echo $this->form->getLabel('published'); ?>
    <?php echo $this->form->getInput('published'); ?></li>

    <li><?php echo $this->form->getLabel('access'); ?>
    <?php echo $this->form->getInput('access'); ?></li>

    <li><?php echo $this->form->getLabel('id'); ?>
    <?php echo $this->form->getInput('id'); ?></li>

    </ul>

    <?php echo $this->form->getLabel('description'); ?>
    <div class="clr"></div>
    <?php echo $this->form->getInput('description'); ?>
  </fieldset>
</div>
```

В этой части исходного кода из файла компоновки формы ввода и редактирования подписок определяется действие для загрузки формы по тому же самому URL, что и

прежде, с идентификатором текущего элемента. Затем метод отправки формы, а также имя, идентификатор и класс определяются в соответствующих атрибутах элемента разметки `form`. Далее первые семь полей формы, размеченные элементами `form` в коде HTML, выводятся на экран в виде неупорядоченного списка. И в каждом случае из XML-файла разметки формы извлекается метка поля и выводится на экран. Эти поля отображаются в левой части формы.

И наконец, ниже приведена последняя часть исходного кода из рассматриваемого здесь файла компоновки формы ввода и редактирования подписок.

```
  <div class="width-40 fltrt">
    <?php echo JHtml::_('sliders.start','joomprosubs-sliders-'.
$this->item->id, array('useCookie'=>1)); ?>

    <?php echo JHtml::_('sliders.panel',JText::_(
'JGLOBAL_FIELDSET_PUBLISHING'), 'publishing-details'); ?>

    <fieldset class="panelform">
      <ul class="adminformlist">
        <li><?php echo $this->form->getLabel('created_by'); ?>
        <?php echo $this->form->getInput('created_by'); ?></li>

        <li><?php echo $this->form->getLabel('created_by_alias'); ?>
        <?php echo $this->form->getInput('created_by_alias'); ?></li>

        <li><?php echo $this->form->getLabel('created'); ?>
        <?php echo $this->form->getInput('created'); ?></li>

        <li><?php echo $this->form->getLabel('publish_up'); ?>
        <?php echo $this->form->getInput('publish_up'); ?></li>

        <li><?php echo $this->form->getLabel('publish_down'); ?>
        <?php echo $this->form->getInput('publish_down'); ?></li>

        <?php if ($this->item->modified_by) : ?>
          <li><?php echo $this->form->getLabel('modified_by'); ?>
          <?php echo $this->form->getInput('modified_by'); ?></li>

          <li><?php echo $this->form->getLabel('modified'); ?>
          <?php echo $this->form->getInput('modified'); ?></li>
        <?php endif; ?>

      </ul>
    </fieldset>

    <?php echo JHtml::_('sliders.end'); ?>

    <input type="hidden" name="task" value="" />
     <?php echo JHtml::_('form.token'); ?>
  </div>
  <div class="clr"></div>
</form>
```

В этой части исходного кода из файла компоновки формы ввода и редактирования подписок определяется панель со слайдерами для отображения сведений о публикации. Для создания этой панели используются методы `JHtmlSliders::start()` и `JHtmlSliders::panel()`. Затем с помощью методов `getLabel()` и `getInput()` на экран выводятся еще семь полей формы для ввода сведений о публикации. Далее создание панели со слайдерами завершается и добавляется скрытое поле с маркером формы. Напомним, что это поле служит для проверки маркера, сохраненного в текущем сеансе работы, чтобы убедиться в достоверности сохраняемой информации.

А теперь рассмотрим XML-файл `models/forms/subscription.xml`, из которого выбиралась разметка полей рассматриваемой здесь формы. Ниже приведена первая часть этого файла.

```
<?xml version="1.0" encoding="utf- 8"?>
<form>
  <fieldset>
    <field name="id" type="text" default="0"
↳label="JGLOBAL_FIELD_ID_LABEL"
      readonly="true" class="readonly"
      description="JGLOBAL_FIELD_ID_DESC"/>

    <field name="title" type="text" class="inputbox"
      size="40" label="JGLOBAL_TITLE"
      description="COM_JOOMPROSUBS_FIELD_TITLE_DESC" required="true" />

    <field name="alias" type="text" class="inputbox"
      size="40" label="JFIELD_ALIAS_LABEL"
     description="COM_JOOMPROSUBS_FIELD_ALIAS_DESC" />

    <field name="catid" type="category" extension="com_joomprosubs"
      label="JCATEGORY"
      description="COM_JOOMPROSUBS_FIELD_CATEGORY_DESC"
      class="inputbox" >
    </field>

    <field name="group_id" type="usergroup"
      label="COM_JOOMPROSUBS_FIELD_USERGROUP_LABEL"
      description="COM_JOOMPROSUBS_FIELD_USERGROUP_DESC"
      default="0" size="1" >
    </field>

    <field name="duration" type="integer" filter="integer"
      first="15" last="90" step="15" default="30"
      label="COM_JOOMPROSUBS_FIELD_DURATION_LABEL"
      description="COM_JOOMPROSUBS_FIELD_DURATION_DESC" />

    <field name="description" type="editor" buttons="true"
      hide="pagebreak,readmore"
      class="inputbox"
      filter="safehtml"
        label="JGLOBAL_DESCRIPTION"
        description="COM_JOOMPROSUBS_FIELD_DESCRIPTION_DESC" />
```

В этой части рассматриваемого здесь XML-файла определяются первые семь полей формы ввода и редактирования подписок. В качестве типа для поля идентификатора категории (`catid`) устанавливается категория (`type="category"`), а в качестве расширения указывается имя создаваемого компонента (`extension="com_joomprosubs"`). Это позволяет сформировать раскрывающийся список из категорий подписок. Аналогичным образом в качестве типа для поля идентификатора группы (`group_id`) указывается группа пользователей (`type="usergroup"`).

Для поля срока действия подписки (`duration`) указывается целочисленный (`integer`) тип и фильтр. Такой тип данного поля позволяет указывать верхний и нижний пределы, а также шаг приращения. В данном случае срок действия подписки начинается с 15 дней и постепенно увеличивается до 90 дней с приращением в 15 дней. А целочисленный фильтр этого поля означает, что даже если злоумышленнику и удастся ввести какие-нибудь другие данные в форму, минуя раскрывающийся список, то эти данные все равно будут преобразованы в целочисленные при сохранении в таблице.

И наконец, для поля описания (`description`) в качестве типа указан редактор (`editor`). Это означает, что для редактирования данных в этом поле будет использоваться редактор, выбираемый пользователем по умолчанию. В качестве фильтра для этого поля указывается безопасная разметка (`safehtml`), чтобы отфильтровать введенные пользователем данные и безопасно воспроизвести их на HTML-странице.

Ниже приведена последняя часть рассматриваемого здесь XML-файла.

```
<field
  name="published"
  type="list"
  label="JSTATUS"
  description="COM_JOOMPROSUBS_FIELD_STATE_DESC"
  class="inputbox"
  size="1"
  default="1">
  <option
    value="1">JPUBLISHED</option>
  <option
    value="0">JUNPUBLISHED</option>
  <option
    value="-2">JTRASHED</option>
 </field>

<field name="access" type="accesslevel" label="JFIELD_ACCESS_LABEL"
  description="JFIELD_ACCESS_DESC" class="inputbox" size="1" />

<field name="created" type="calendar"
  label="JGLOBAL_FIELD_CREATED_LABEL"
  description="JGLOBAL_FIELD_CREATED_DESC"
  class="inputbox" size="22" format="%Y-%m-%d %H:%M:%S"
  filter="user_utc" />

<field name="created_by" type="user"
  label="JGLOBAL_FIELD_CREATED_BY_LABEL"
  description="JGLOBAL_FIELD_CREATED_BY_Desc" />
```

```
    <field name="created_by_alias" type="text"
      label="JGLOBAL_FIELD_CREATED_BY_ALIAS_LABEL"
      description="JGLOBAL_FIELD_CREATED_BY_ALIAS_DESC"
      class="inputbox" size="20" />

    <field name="modified" type="calendar" class="readonly"
      label="JGLOBAL_FIELD_MODIFIED_LABEL"
      description="COM_JOOMPROSUBS_FIELD_MODIFIED_DESC"
      size="22" readonly="true" format="%Y-%m-%d %H:%M:%S"
      filter="user_utc" />

    <field name="modified_by" type="user"
      label="JGLOBAL_FIELD_MODIFIED_BY_LABEL"
      class="readonly" readonly="true" filter="unset" />

    <field name="checked_out" type="hidden" filter="unset" />

    <field name="checked_out_time" type="hidden" filter="unset" />

    <field name="publish_up" type="calendar"
      label="JGLOBAL_FIELD_PUBLISH_UP_LABEL"
      description="JGLOBAL_FIELD_PUBLISH_UP_DESC"
      class="inputbox" format="%Y-%m-%d %H:%M:%S" size="22"
      filter="user_utc" />

    <field name="publish_down" type="calendar"
      label="JGLOBAL_FIELD_PUBLISH_DOWN_LABEL"
      description="JGLOBAL_FIELD_PUBLISH_DOWN_DESC"
      class="inputbox" format="%Y-%m-%d %H:%M:%S" size="22"
      filter="user_utc" />

  </fieldset>

</form>
```

В этой части рассматриваемого здесь XML-файла определяются последние семь полей формы ввода и редактирования подписок. Обратите внимание на то, что в качестве типа для полей дат указывается календарь (`calendar`), чтобы предоставить доступ к всплывающей панели с календарем. Здесь также вводятся в форму два скрытых поля для хранения признака снятия с регистрации (`checked_out`) и времени снятия с регистрации (`checked_out_time`). Ввести данные в эти поля нельзя непосредственно в форме, поскольку в их атрибуте `filter` установлено значение `"unset"`. Эти поля специально сделаны недоступными в форме, чтобы злоумышленник не смог ввести в них злонамеренные данные.

Табличный класс

И наконец, рассмотрим последний файл, используемый в процессе ввода и редактирования подписок. Это файл `tables/subscription.php`, в котором объявляется класс `JoomprosubsTableSubscription`. В нем организуется специальная обработка, необходимая для сохранения строки в таблице базы данных. Ниже приведена первая часть исходного кода из этого файла.

```
defined('_JEXEC') or die;
class JoomprosubsTableSubscription extends JTable
{
/**
 * Конструктор класса.
 *
 * @param   Jdatabase   Объект, соединяющий с базой данных.
 */
  public function __construct(&$db)
  {
    parent::__construct('#__joompro_subscriptions', 'id', $db);
  }
```

В этой части исходного кода из рассматриваемого здесь файла объявляется класс, а таблице базы данных присваивается имя, передаваемое конструктору данного класса в качестве первого аргумента. Обратите внимание на использование знака амперсанда (**&**) перед именем аргумента. Как упоминалось ранее, это означает, что при внесении изменений в объект `$db` в теле конструктора данного класса измененный объект будет передан обратно вызывающей части программы.

В следующей части рассматриваемого здесь файла определяется метод `store()`, как показано ниже.

```
/**
 * Перегружаемый метод сохранения данных в таблице подписок.
 *
 * @param   boolean          Обозначает, следует ли обновить пустые значения.
 * @return  boolean          Истинно при успешном завершении, а иначе – ложно.
 */
public function store($updateNulls = false)
{
  $date = JFactory::getDate();
  $user = JFactory::getUser();
  if ($this->id) {
    // Существующий элемент.
    $this->modified = $date->toSQL();
    $this->modified_by = $user->get('id');
  } else {
    // Новая подписка. Поля created и created_by field могут быть
    // установлены пользователем, и поэтому их нельзя трогать,
    // если они уже установлены.
    if (!intval($this->created)) {
      $this->created = $date->toSQL();
    }
    if (empty($this->created_by)) {
      $this->created_by = $user->get('id');
    }
  }

  // проверить однозначность псевдонима
  $table = JTable::getInstance('subscription', 'JoomprosubsTable');
  if ($table->load(array('alias'=>$this->alias,'catid'=>$this->catid))
&& ($table->id != $this->id || $this->id==0)) {
    $this->setError(JText::_('COM_JOOMPROSUBS_ERROR_UNIQUE_ALIAS'));
```

```
    return false;
  }
  // попытаться сохранить введенные пользователем данные
  return parent::store($updateNulls);
}
```

Этот метод вызывается при сохранении строки в таблице базы данных. Если существующая строка редактируется, то в поле видоизмененной даты (`modified`) устанавливается текущая дата, а в поле видоизменений (`modified_by`) — текущий пользователь. Если же создается новая строка, то в полях `created` и `created_ by` устанавливаются текущая дата и пользователь соответственно. Затем в данном методе проверяется однозначность псевдонима данной категории. Это делается с целью исключить недоразумения на тот случай, если пользователь неумышленно создаст в одной и той же категории два элемента содержимого с одинаковым псевдонимом. И если нечто подобное произойдет, то в пользовательской части веб-сайта может быть отображен лишь один элемент содержимого, поскольку оба элемента имеют одинаковый URL. Именно этому и воспрепятствует данная проверка, сообщая пользователю об ошибке.

И в конце данного метода из родительского класса вызывается метод `store()` для выполнения стандартной операции сохранения данных в таблице. В эту операцию входит установка поля ключа и обновление таблицы ресурсов по мере надобности.

И в последней части рассматриваемого здесь файла определяется метод `check()`, как показано ниже.

```
/**
 * Перегружаемый метод проверки целостности данных.
 *
 * @return            boolean          Истинно при успешном завершении.
 */
public function check()
{
  // проверить существующее имя
  $db = $this->_db;
  $query = $db->getQuery(true);
  $query->select('id');
  $query->from($db->quoteName('#__joompro_subscriptions'));
  $query->where('title = ' . $db->quote($this->title) .
      ' AND catid = ' . (int) $this->catid);
  $db->setQuery($query);

  $xid = intval($db->loadResult());
  if ($xid && $xid != intval($this->id)) {
    $this->setError(JText::_('COM_JOOMPROSUBS_ERR_TABLES_NAME'));
    return false;
  }

  if (empty($this->alias)) {
    $this->alias = $this->title;
  }

  $this->alias = JApplication::stringURLSafe($this->alias);
  if (trim(str_replace('- ','',$this- >alias)) == '') {
```

```
      $this->alias = JFactory::getDate()->format("Y-m-d-H-i-s");
    }

    // проверить, не является ли конечная дата публикации более ранней,
    // чем начальная дата публикации
    if (intval($this->publish_down) > 0 && $this->publish_down <
↳$this->publish_up) {
      // поменять даты
      $temp = $this->publish_up;
      $this->publish_up = $this->publish_down;
      $this->publish_down = $temp;
    }

    return true;
  }
}
```

И в этом методе проверяется однозначность имени подписки для данной категории, чтобы исключить дублирование значений псевдонимов. Кроме того, проверяется, не является ли конечная дата публикации подписки более ранней, чем начальная дата. Таким образом, мы определили все файлы, требующиеся для ввода и редактирования подписок в административной части рассматриваемого здесь компонента.

Файлы языковой поддержки

Для административной части рассматриваемого здесь компонента требуются два файла языковой поддержки. Эти файлы должны храниться в папке языковой поддержки данного компонента (`administrator/components/com_joomprosubs/language/en-GB/`), а не в общей папке `administrator/language`. Первым из них является основной файл `language/en-GB/en-GB.com_joomprosubs.ini`, который содержит все переводы для языковых ключей, используемых в данном компоненте. Ниже приведено содержимое этого файла.

```
COM_JOOMPROSUBS_CATEGORY_LIST_HELP_LINK="http://joomlaprogrammingbook.com/
↳joompro-category-list-help.html"
COM_JOOMPROSUBS_CATEGORY_VIEW_DEFAULT_TITLE="Category List"
COM_JOOMPROSUBS_CONFIGURATION="Subscriptions Manager Options"
COM_JOOMPROSUBS_EDIT_JOOMPROSUB="Subscription Edit"
COM_JOOMPROSUBS_FIELD_ALIAS_DESC="Alias for the item. You can leave this
↳blank and the system will create an alias for you."
COM_JOOMPROSUBS_FIELD_CATEGORY_DESC="Select the category from the list."
COM_JOOMPROSUBS_FIELD_DESCRIPTION_DESC="Enter an optional description for
↳the subscription."
COM_JOOMPROSUBS_FIELD_DURATION_DESC="Select the number of days that this
↳subscription will be active."
COM_JOOMPROSUBS_FIELD_DURATION_LABEL="Duration (days)"
COM_JOOMPROSUBS_FIELD_MODIFIED_DESC="Date this subscription was last
↳modified."
COM_JOOMPROSUBS_FIELD_SELECT_CATEGORY_DESC="Select the category from the
↳list."
COM_JOOMPROSUBS_FIELD_SELECT_CATEGORY_LABEL="Category"
```

```
COM_JOOMPROSUBS_FIELD_STATE_DESC="Published state for this subscription."
COM_JOOMPROSUBS_FIELD_TITLE_DESC="Title of the subscription."
COM_JOOMPROSUBS_FIELD_USERGROUP_DESC="Select the User Group that is
↳associated with this subscription."
COM_JOOMPROSUBS_FIELD_USERGROUP_LABEL="User Group"
COM_JOOMPROSUBS_MANAGER_JOOMPROSUB="Subscription Manager: Subscription
↳Entry"
COM_JOOMPROSUBS_MANAGER_JOOMPROSUBS="Subscriptions Manager: Subscriptions"
COM_JOOMPROSUBS_N_ITEMS_ARCHIVED_1="%d subscription successfully archived"
COM_JOOMPROSUBS_N_ITEMS_ARCHIVED="%d subscriptions successfully archived"
COM_JOOMPROSUBS_N_ITEMS_CHECKED_IN_0="No subscription successfully checked
↳in"
COM_JOOMPROSUBS_N_ITEMS_CHECKED_IN_1="%d subscription successfully checked
↳in"
COM_JOOMPROSUBS_N_ITEMS_CHECKED_IN_MORE="%d subscriptions successfully
↳checked in"
COM_JOOMPROSUBS_N_ITEMS_DELETED_1="%d subscription successfully deleted"
COM_JOOMPROSUBS_N_ITEMS_DELETED="%d subscriptions successfully deleted"
COM_JOOMPROSUBS_N_ITEMS_PUBLISHED_1="%d subscription successfully
↳published"
COM_JOOMPROSUBS_N_ITEMS_PUBLISHED="%d subscriptions successfully
↳published"
COM_JOOMPROSUBS_N_ITEMS_TRASHED_1="%d subscription successfully trashed"
COM_JOOMPROSUBS_N_ITEMS_TRASHED="%d subscriptions successfully trashed"
COM_JOOMPROSUBS_N_ITEMS_UNPUBLISHED_1="%d subscription successfully
↳unpublished"
COM_JOOMPROSUBS_N_ITEMS_UNPUBLISHED="%d subscriptions successfully
↳unpublished"
COM_JOOMPROSUBS_NEW_JOOMPROSUB="Subscription Add"
COM_JOOMPROSUBS_SEARCH_IN_TITLE="Searh in title."
COM_JOOMPROSUBS_SUBMANAGER_HELP_LINK="http://joomlaprogrammingbook.com/
↳joompro-subscription-manager-help.html"
COM_JOOMPROSUBS_SUBMENU_CATEGORIES="Categories"
COM_JOOMPROSUBS_SUBMENU_JOOMPROSUBS="Subscriptions"
COM_JOOMPROSUBS_SUBSCRIPTION_HELP_LINK="http://joomlaprogrammingbook.com/
↳joompro-subscription-edit-help.html"
COM_JOOMPROSUBS_TOOLBAR_CSVREPORT="Report"
COM_JOOMPROSUBS="Subscriptions"
```

Вторым является системный файл `language/en-GB/en-GB.com_joomprosubs.sys.ini`, содержащий языковые символьные строки, требующиеся за пределами данного компонента. Ниже приведено содержимое этого файла.

```
CATEGORIES="Categories"
COM_JOOMPROSUBS="JoomPro Subscriptions"
COM_JOOMPROSUBS_XML_DESCRIPTION="This is an example component for the
↳Joomla! Programming book."
COM_JOOMPROSUBS_CATEGORY_VIEW_DEFAULT_DESC="Lists all subscriptions in a
↳category."
COM_JOOMPROSUBS_CATEGORY_VIEW_DEFAULT_OPTION="Default Layout"
COM_JOOMPROSUBS_CATEGORY_VIEW_DEFAULT_TITLE="Category List"
```

Установка и настройка компонента

И последним в административной части рассматриваемого здесь компонента является установочный файл joomprosubs.xml, находящийся в папке administrator/components/com_joomprosubs. Этот файл будет построен в два этапа. Сначала в него будут включены файлы только из административной части, что позволит установить и проверить данную часть компонента, прежде чем переходить к разработке пользовательской его части. А в следующей главе в этот установочный файл будут включены файлы из пользовательской части данного компонента.

Ниже приведена первая часть этого установочного файла.

```
<?xml version="1.0" encoding="utf-8"?>
<extension type="component" version="2.5.0" method="upgrade">
    <name>com_joomprosubs</name>
    <author>Mark Dexter and Louis Landry</author>
    <creationDate>January 2012</creationDate>
    <copyright>(C) 2012 Mark Dexter and Louis Landry. All rights reserved.
    </copyright>
    <license>GNU General Public License version 2 or later; see
       LICENSE.txt</license>
    <authorEmail>admin@joomla.org</authorEmail>
    <authorUrl>www.joomla.org</authorUrl>
    <version>2.5.0</version>
    <description>COM_JOOMPROSUBS_XML_DESCRIPTION</description>

    <install> <!-- Выполняется при установке компонента -->
        <sql>
          <file driver="mysql"
↳charset="utf8">sql/install.mysql.utf8.sql</file>
        </sql>
    </install>
    <uninstall> <!-- Выполняется при удалении компонента -->
        <sql>
          <file driver="mysql"
↳charset="utf8">sql/uninstall.mysql.utf8.sql</file>
        </sql>
    </uninstall>
```

Сначала в этой части установочного XML-файла определяются элементы разметки с описанием имени компонента, его автора и так далее вплоть до элемента description, а затем определяются элементы разметки установки и удаления компонента. В этих элементах указываются имена сценариев базы данных, которые должны выполняться при установке и удалении данного компонента. В данном примере по этим сценариям создаются и удаляются соответственно две таблицы, предназначенные для рассматриваемого здесь компонента.

Ниже приведена следующая часть установочного XML-файла.

```
<administration>
  <menu img="class:newsfeeds">COM_JOOMPROSUBS</menu>
  <submenu>
    <!--
```

```
    Примечание: все знаки & должны быть экранированы следующим
    образом: &amp, чтобы XML-файл оказался действительным и мог
    быть подвергнут синтаксическому анализу в программе установки
  -->
  <menu link="option=com_joomprosubs" view="submanager"
img="class:newsfeeds"
      alt="Subscriptions/Subscriptions">COM_JOOMPROSUBS</menu>
  <menu link="option=com_categories&amp;extension=com_joomprosubs"
      view="categories" img="class:newsfeeds-cat"
      alt="Subscriptions/Categories">Categories</menu>
 </submenu>
```

Эта часть рассматриваемого здесь установочного XML-файла начинается с элемента разметки `administration`, где определяются меню и файлы административной части данного компонента. Они практически ничем не отличаются от тех, что были рассмотрены в главе 7 для компонента Weblinks, за исключением того, что в них используется имя другого, рассматриваемого здесь компонента. Здесь вводятся один пункт меню верхнего уровня и два подменю для выбора данного компонента, как показано на рис. 9.4.

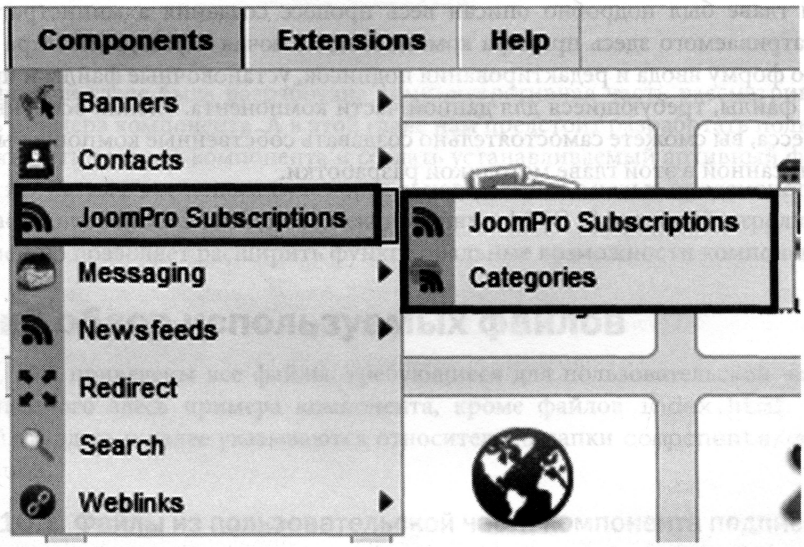


Рис. 9.4. Пункты меню Subscriptions

И последняя часть установочного XML-файла приведена ниже.

```
<files folder="admin">
    <folder>controllers</folder>
    <folder>helpers</folder>
    <folder>language</folder>
    <folder>models</folder>
    <folder>sql</folder>
    <folder>tables</folder>
    <folder>views</folder>
    <filename>access.xml</filename>
```

```
        <filename>config.xml</filename>
        <filename>controller.php</filename>
        <filename>index.html</filename>
        <filename>joomprosubs.php</filename>
    </files>
  </administration>
</extension>
```

В этой части установочного XML-файла определяются файлы для административной части рассматриваемого здесь компонента. Здесь, опять же, определяются только папки и файлы верхнего уровня. А затем закрываются элементы разметки `administration` и `extension`.

А теперь можно перейти по команде Extension Manager⇨Discover и воспользоваться пиктограммами Discover и Install на панели инструментов экрана Discover, чтобы установить и проверить административную часть рассматриваемого здесь компонента.

Резюме

В этой главе был подробно описан весь процесс создания административной части рассматриваемого здесь примера компонента, включая экран диспетчера подписок, экраннную форму ввода и редактирования подписок, установочные файлы и прочие разнотипные файлы, требующиеся для данной части компонента. Уяснив особенности данного процесса, вы сможете самостоятельно создавать собственные компоненты, руководствуясь описанной в этой главе методикой разработки.

10

Компоненты, часть IV. Пример разработки пользовательской части компонента

В предыдущей главе была разработана административная часть рассматриваемого здесь примера компонента. А в этой главе нам предстоит разработать пользовательскую часть данного компонента и создать устанавливаемый архивный файл в формате zip. Затем в компонент будет введен новый отчет с целью продемонстрировать, насколько просто шаблон проектирования MVC (Модель–Контроллер–Представление) позволяет расширить функциональные возможности компонента.

Краткий обзор используемых файлов

В табл. 10.1 приведены все файлы, требующиеся для пользовательской части рассматриваемого здесь примера компонента, кроме файлов `index.html`. Все имена файлов здесь и далее указываются относительно папки `components/com_joomprosubs`.

Таблица 10.1. Файлы из пользовательской части компонента подписок

Имя файла	Содержимое
`controller.php`	Класс `JoomproSubsController`
`joomprosubs.php`	Сценарий, служащий в качестве точки входа в компонент
`controllers/subscription.php`	Класс `JoomproSubsControllerSubscription`
`helpers/category.php`	Класс `JoomproSubsCategories`
`language/en-GB/en-GB.com_joomprosubs.ini`	Файл языковой поддержки
`models/category.php`	Класс `JoomprosubsModelCategory` (для пункта меню единственной категории)

Окончание табл. 10.1

Имя файла	Содержимое
models/form.php	Класс JoomprosubsModelForm (для формы регистрации на подписку)
models/forms/subscription.xml	XML-файл для разметки формы
views/category/view.html.php	Класс JoomprosubsViewCategory (для пункта меню единственной категории)
views/category/tmpl/default.php	Компоновка пункта меню единственной категории
views/category/tmpl/default.xml	XML-файл для разметки параметров пункта меню единственной категории
views/category/tmpl/default_items.php	Файл подчиненной компоновки пункта меню единственной категории
views/form/view.html.php	Класс JoomprosubsViewForm (для формы предъявления подписки)
views/form/tmpl/edit.php	Файл компоновки формы предъявления подписки
views/form/tmpl/thankyou.php	Файл компоновки сообщения, благодарящего пользователя за подписку

Установочный XML-файл

В предыдущей главе был создан установочный файл administrator/components/com_joomprosubs/joomprosubs.xml, содержащий сведения о файлах из административной части рассматриваемого здесь примера компонента. Его создание было остановлено на первой стадии, чтобы можно было установить и проверить административную часть компонента, и поэтому в него не вошли сведения о файлах из пользовательской части данного компонента. В этой главе будет продолжено создание установочного файла, чтобы дополнить его недостающими сведениями о файлах из пользовательской части.

Итак, начнем вводить файлы из пользовательской части в установочный XML-файл, как показано ниже.

```
. . .
</uninstall>
<files folder="site">
  <folder>controllers</folder>
  <folder>helpers</folder>
  <folder>language</folder>
  <folder>models</folder>
  <folder>views</folder>
  <filename>controller.php</filename>
  <filename>index.html</filename>
  <filename>joomprosubs.php</filename>
</files>
<administration>
. . .
```

Выделенные полужирным строки кода являются новыми в данном файле. В них вводится элемент разметки files между элементами разметки uninstall и administration. Как и прежде, в элементе разметки files перечисляются только файлы верхнего уровня

из файловой структуры компонента. Присваивание значения "site" атрибуту folder означает, что установочная программа должна скопировать указанные далее папки в папку components пользовательской части компонента.

Точка входа в компонент

Прежде всего необходимо создать папку components/com_joomprosubs для пользовательской части рассматриваемого здесь компонента. В этой папке должны храниться все файлы из пользовательской части данного компонента, и поэтому они будут далее указываться относительно ее.

В качестве точки входа в пользовательскую часть данного компонента служит файл joomprosubs.php. Как и следовало ожидать, он содержит следующий исходный код:

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

jimport('joomla.application.component.controller');
$controller = JController::getInstance('Joomprosubs');
$controller->execute(JRequest::getCmd('task'));
$controller->redirect();
```

Исходный контроллер

Исходный контроллер, обрабатывающий задачу отображения, определяется в классе JoomproSubsController из файла controller.php. Ниже приведен исходный код из этого файла.

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

jimport('joomla.application.component.controller');

/**
 * Контроллер компонента Joomprosubs.
 *
 */
class JoomproSubsController extends JController
{
  /**
   * Метод для отображения представления.
   *
   * @param       boolean          Если истинно, то выводимый из представления
   *                               результат будет кеширован.
   * @param       array       Массив параметров безопасного URL и типов
   *                          их переменных, а действительные значения
   *                          см. в {@link JFilterInput::clean()}.
   *
   * @return            JController Этот объект поддерживает связывание
   *                               методов в цепочку.
   */
  public function display($cachable = false, $urlparams = false)
  {
    // инициализировать переменные
    $cachable = true;
```

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

    // Задать имя и формат представления, извлекаемые по умолчанию из
    // запроса. Во избежание конфликтов с маршрутизатором и возвращаемой
    // страницей здесь используется обозначение идентификатора sub_id.
    $id = JRequest::getInt('sub_id');
    $vName = JRequest::getCmd('view', 'category');
    JRequest::setVar('view', $vName);

    if ($user->get('id')) {
      $cachable = false;
    }

    $safeurlparams = array(
      'id'                          => 'INT',
      'limit'                       => 'INT',
      'limitstart'                  => 'INT',
      'filter_order'                => 'CMD',
      'filter_order_Dir'            => 'CMD',
      'lang'                             => 'CMD'
    );

    // проверить наличие формы редактирования
    if ($vName == 'form' && !$this->checkEditId(
↳'com_joomprosubs.edit.subscription', $id)) {
      // не разрешать несанкционированный доступ к форме
      return JError::raiseError(403,
↳JText::sprintf('JLIB_APPLICATION_ERROR_UNHELD_ID', $id));
    }

    return parent::display($cachable,$safeurlparams);
  }
}
```

Этот класс практически ничем не отличается от класса исходного контроллера в пользовательской части компонента Weblinks. Сначала в его методе `display()` определяется необходимость кешировать представление, т.е. сохранять предыдущую его копию в оперативно загружаемом кеш-файле. Если пользователь регистрируется, то вряд ли стоит пытаться использовать кеш-файл. Ведь в таком случае придется проверять базу данных всякий раз, когда выясняются полномочия пользователя на подписку, а для этого кеширование не годится.

Затем получаются идентификатор подписки и имя представления. Если представление не указано в запросе, то по умолчанию выбирается представление категории. В данном примере имеется лишь одно такое представление под названием `category`. Кроме того, создается массив `$safeurlparams`, содержащий все действительные параметры для URL. А все остальные параметры в URL являются недействительными, что дает возможность удалить любой недостоверный код из URL.

И наконец, как и прежде, из метода `display()` данного класса возвращается результат выполнения аналогичного метода из родительского класса (в данном случае — `JController`). Этот метод возвращает переменную `$this`. Как пояснялось ранее, возврат переменной `$this` позволяет связывать методы в цепочку.

Представление категории подписок

В пользовательской части данного компонента имеется единственное представление — категории. Оно подобно представлениям списков основных категорий. В нем отображаются все подписки из единственной категории. Рассмотрим сначала, каким образом создается пункт меню для выбора этого представления.

XML-файл для разметки пункта меню

Для каждого пункта предоставляется ряд вариантов выбора, называемых параметрами, управляющими его отображением. Как и все остальные параметры, они указываются в XML-файле. Этот файл находится в папке `tmpl` представления и называется таким же образом, как и файл компоновки данного пункта меню, а отличается он него лишь своим расширением `xml`. В данном случае этот файл называется `views/category/tmpl/default.xml`. Ниже приведено его содержимое.

```
<?xml version="1.0" encoding="utf-8"?>
<metadata>
  <layout title="COM_JOOMPROSUBS_CATEGORY_VIEW_DEFAULT_TITLE"
    option="COM_JOOMPROSUBS_CATEGORY_VIEW_DEFAULT_OPTION">
    <help url="COM_JOOMPROSUBS_CATEGORY_LIST_HELP_LINK"
    />
    <message>
      <![CDATA[COM_JOOMPROSUBS_CATEGORY_VIEW_DEFAULT_DESC]]>
    </message>
  </layout>

  <!-- Дополнить переменные запроса полями для компоновки. -->
  <fields name="request">
    <fieldset name="request">
      <field name="id" type="category"
        default="0"
        description="COM_JOOMPROSUBS_FIELD_SELECT_CATEGORY_DESC"
        extension="com_joomprosubs"
        label="COM_JOOMPROSUBS_FIELD_SELECT_CATEGORY_LABEL"
        required="true"
      />
    </fieldset>
  </fields>

  <!-- Дополнить объект параметров полями для компоновки. -->
  <fields name="params">
    <fieldset name="basic" label="JGLOBAL_CATEGORY_OPTIONS">
      <field name="show_description" type="list"
        description="JGLOBAL_SHOW_CATEGORY_DESCRIPTION_DESC"
        label="JGLOBAL_SHOW_CATEGORY_DESCRIPTION_LABEL"
      >

        <option value="0">JHIDE</option>
        <option value="1">JSHOW</option>
      </field>
    </fieldset>
  </fields>
</metadata>
```

В элементе разметки `layout` указываются название и описание, которые будут отображаться вместе со списком доступных пунктов меню. И в этом случае для обращения к справочному файлу используется URL. Так, если щелкнуть на пиктограмме Help непосредственно на панели инструментов во время ввода или редактирования пункта меню, предназначенного для выбора категории подписок из списка, то в новом окне браузера будет загружен справочный файл по URL, указанному в файле языковой поддержки. Это дает возможность создавать справочные системы веб-сайтов на разных языках.

Далее следуют два элемента разметки `fields`. Первый из них называется `request` и содержит одноименный элемент разметки `fieldset`. Этот элемент будет добавлен в переменную запроса при загрузке данного пункта меню. Как правило, в разметке пункта меню имеется лишь одно поле запроса, например, поле идентификатора категории или статьи, но в данном случае дополнительно указываются тип `category` и расширение `com_joomprosubs`. Поэтому для предоставления пользователю раскрывающегося списка со всеми категориями данного компонента используется базовый класс `JFormFieldCategory` из файла `libraries/joomla/form/fields/category.php`.

В папке `libraries/joomla/form/fields` содержатся файлы классов, поддерживающие предопределенные типы полей из класса `JForm`. В главе 6 был создан класс `JFormRule`, который вызывался при установке значения, совпадавшего с именем файла, где этот класс был объявлен, в атрибуте `validate` элемента `field` разметки поля из класса `JForm`. Аналогичным образом можно создать и специальные классы типа `JFormField`. Для этого достаточно ввести в элемент `fieldset` разметки поля из класса `JForm` атрибут `addfieldpath`, указывающий на папку специального класса. При этом значение атрибута `type` элемента разметки `field` должно совпадать с именем файла, в котором объявлен специальный класс типа `JFormField`. Такой прием неоднократно применяется в Joomla, например, в файле `administrator/components/com_content/config.xml`.

Возвращаясь к рассматриваемому здесь XML-файлу, отметим, что в нем имеется еще один элемент разметки `fields` под названием `params`. Этот элемент содержит один элемент разметки `fieldset` под названием `basic` и один элемент разметки `field`, называемый `show_description`. Поля этих параметров размечаются точно так же, как и рассмотренные выше поля. В них пользователю предоставляется возможность отобразить описание категории. На рис. 10.1 приведены моментальные снимки экрана с двумя параметрами, выбираемыми при создании нового пункта меню, предназначенного для выбора категории подписок из списка.

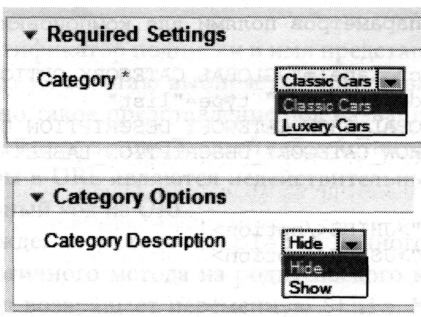


Рис. 10.1. Параметры пункта меню, показанные в развернутом виде

Представление категории

Рассмотрим далее класс представления JoomproSubsViewCategory. Этот класс объявляется в файле category/view.html.php. Ниже приведена первая часть исходного кода из этого файла.

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

jimport('joomla.application.component.view');
/**
 * Класс HTML-представления для компонента JoomproSubs
 *
 */
class JoomprosubsViewCategory extends JView
{
  protected $state;
  protected $items;
  protected $category;
  protected $children;
  protected $pagination;

  function display($tpl = null)
{
    $app = JFactory::getApplication();
    $params = $app- >getParams();

    // получить данные из моделей
    $state = $this->get('State');
    $items = $this->get('Items');
    $category = $this->get('Category');
    $pagination = $this->get('Pagination');

    // проверить ошибки
    if (count($errors = $this->get('Errors'))) {
      JError::raiseError(500, implode("\n", $errors));
      return false;
    }

    if ($category == false) {
      return JError::raiseWarning(404,
↳JText::_('JGLOBAL_CATEGORY_NOT_FOUND'));
    }
```

Этот код аналогичен коду представления категории из компонента Weblinks. В нем определяется ряд полей для использования в классе JoomproSubsViewCategory и его методе display(). Напомним, что этот метод вызывается из метода display() в классе контроллера. Сначала в этом методе из модели получаются состояние, список элементов, категория и разбиение на страницы, а затем проверяются ошибки.

Ниже приведена следующая часть исходного кода метода display().

```
// проверить, является ли уровень доступа подходящим для категории
$user = JFactory::getUser();
$groups = $user->getAuthorisedViewLevels();
```

```
if (!in_array($category->access, $groups)) {
  return JError::raiseError(403, JText::_('JERROR_ALERTNOAUTHOR'));
}

// подготовить данные
// заготовить элементы подписок для формирования URL
for ($i = 0, $n = count($items); $i < $n; $i++)
{
  $item = &$items[$i];
  $item->slug = $item->alias ? ($item->id.':'.$item->alias) : $item->id;
}

// установить параметры категории
$cparams = $category->getParams();
$category->params = clone($params);
$category->params->merge($cparams);

$this->state = $state;
$this->items = $items;
$this->category = $category;
$this->params = $params;
$this->pagination = $pagination;

// экранировать символьные строки для вывода в формате HTML
$this->pageclass_sfx = htmlspecialchars($params->get('pageclass_sfx'));
```

Сначала в приведенном выше фрагменте кода проверяется, разрешено ли пользователю просматривать данную категорию. Для получения массива уровней доступа к категориям вызывается метод `AuthorisedViewLevels()` из класса `JUser`, а затем в этом массиве проверяется наличие уровня доступа к данной категории для ее просмотра. Если такой уровень отсутствует, выводится сообщение с кодом ошибки **403**.

Далее следует цикл, где рассчитывается "заготовочное" значение каждого элемента подписки. Это значение составляется из идентификатора элемента, двоеточия и псевдонима. После этого устанавливаются параметры категории, которые затем объединяются с параметрами пункта меню. В данном примере специальные параметры категории отсутствуют, но администратору веб-сайта можно предоставить возможность указывать параметры на уровне категории и компонента, а затем объединять их с параметрами данного пункта меню.

Рассмотрим подробнее назначение метода `merge()` из класса `JRegistry`. Ведь этот метод очень важен для понимания того, каким образом иерархические параметры реализуются в Joomla. В данном примере этой цели служит следующий код:

```
$cparams = $category->getParams();
$category->params = clone($params);
$category->params->merge($cparams);
```

В первой строке этого кода параметры, устанавливаемые на уровне категории, помещаются в переменной `$cparams`. Во второй строке кода вызывается функция `clone()` языка PHP для создания новой копии параметров на текущем уровне компонента, а затем эта копия сохраняется в поле `params` объекта типа `JCategory`. Функция `clone()` использу-

ется для того, чтобы не изменять исходный объект в переменной $params. А в последней строке кода происходит объединение параметров.

При вызове метода merge() из класса JRegistry указывается вызывающий объект (в данном случае — $category->params) и объединяемый объект ($cparams). Логика объединения обоих объектов такова.

- Если значение установлено в одном объекте и не установлено (или не определено) в другом, то используется установленное значение.
- Если значение установлено в обоих объектах, то приоритет отдается значению в вызывающем объекте по сравнению со значением в объединяемом объекте, указываемом в качестве аргумента данного метода.

В базовых компонентах подобным образом наследуются глобальные параметры, устанавливаемые на нижних уровнях иерархии, причем значение "Use Global" (Использовать глобальный параметр) равнозначно пустому или неустановленному значению. Так, если у пункта меню имеются параметры, они получают приоритет над параметрами, устанавливаемыми на уровне компонента. Когда объединяются объекты типа JRegistry, вызывающим оказывает объект типа JRegistry, используемый для хранения параметров пункта меню, а объединяемым — объект, используемый на уровне компонента. И если у пункта меню имеется пустое значение, что равнозначно значению "Use Global", то для объединяемого объекта будет выбрано значение, установленное на уровне компонента. А если значение установлено в обоих объектах, то используется значение, заданное для пункта меню, поскольку он является вызывающим объектом.

Далее в рассматриваемом здесь коде устанавливаются поля класса, в которых сохраняется состояние, список элементов, категория, параметры и разбиение на страницы, чтобы сделать их доступными из сценариев компоновки. После этого вызывается функция htmlspecialchars() с целью экранировать добавляемый суффикс класса страницы и тем самым защитить от любых попыток внесения злонамеренного кода.

Ниже приведена последняя часть исходного кода из метода display().

```
  // проверить наличие замещения компоновки только в том случае,
  // если это не активный пункт меню, а иначе идентификаторы категории
  // и представления должны совпасть
  $active = $app->getMenu()->getActive();
  if ((!$active) || (strpos($active->link, 'view=category') === false)) {
    if ($layout = $category->params->get('category_layout')) {
      $this->setLayout($layout);
    }
  }
  elseif (isset($active->query['layout'])) {
    // задать компоновку на тот случай, если это альтернативный
    // пункт меню с альтернативной компоновкой
    $this->setLayout($active->query['layout']);
  }

  $this->_prepareDocument();

  parent::display($tpl);
}
```

В приведенном выше фрагменте кода сначала проверяется наличие файлов альтернативных компоновок. И хотя в данном примере особой необходимости в такой проверке нет, тем не менее следует иметь в виду, что она производится в том случае, если для категории допускается несколько возможных компоновок и если делается ссылка на компоновку категории, не связанную с конкретным пунктом меню. В таком случае нужно знать, какой именно компоновкой пользоваться для категории. Компоновка может быть задана на уровне компонента или категории. Так, если перейти по команде Article Manager⇨Options⇨Category (Диспетчер статей⇨Параметры⇨Категория), можно обнаружить параметр Choose Layout (Выбрать компоновку), позволяющий выбрать по умолчанию компоновку блога (Blog) или списка (List) статей.

Логика выполнения рассматриваемого здесь кода следующая.

- Если пункт меню активизирован, значит, он существует.
- Если представление категории не обрабатывается, то и хорошо.
- Если же пункт меню не активизирован, но представление категории обрабатывается, то для данной категории выбирается и применяется альтернативная компоновка.
- В противном случае проверяется, указана ли компоновка в запросе из URL. И если она указана, то применяется для данной категории.

После обработки логики выбора альтернативной компоновки сначала вызывается метод `_prepareDocument()`, а затем метод `display()` из родительского класса. Метод `_prepareDocument()` похож на соответствующий метод из компонента Weblinks. Ниже приведена первая часть исходного кода из этого метода.

```
protected function _prepareDocument()
{
  $app = JFactory::getApplication();
  $menu = $app->getMenu()->getActive();
  $pathway = $app->getPathway();
  $title = null;

  if ($menu) {
    $this->params->def('page_heading', $this->params->get
↳('page_title', $menu->title));
  }
  else {
    $this->params->def('page_heading',
↳JText::_('COM_JOOMPROSUBS_DEFAULT_PAGE_TITLE'));
  }

  $title = $this->params->get('page_title', '');
  if (empty($title)) {
    $title = $app->getCfg('sitename');
  }
  elseif ($app->getCfg('sitename_pagetitles', 0)) {
    $title = JText::sprintf('JPAGETITLE', $app->getCfg('sitename'),
↳$title);
  }
  elseif ($app->getCfg('sitename_pagetitles', 0) == 2) {
    $title = JText::sprintf('JPAGETITLE', $title,
```

```
↳$app->getCfg('sitename'));
  }

$this->document->setTitle($title);
```

В приведенном выше фрагменте кода заголовок страницы формируется по иерархии, начиная с заглавия меню, исходного названия компонента, а далее — наименования вебсайта и некоторых данных глобальной настройки. Самое главное — определить как можно более конкретный, но наполненный определенным содержанием заголовок страницы.

Ниже приведена остальная часть исходного кода из метода `_prepareDocument()` и его класса.

```
    if ($this->category->metadesc) {
      $this->document->setDescription($this->category->metadesc);
    }
    elseif (!$this->category->metadesc && $this->params->get(
↳'menu-meta_description')) {
      $this->document->setDescription($this->params->get(
↳'menu-meta_description'));
    }

    if ($this->category->metakey) {
      $this->document->setMetadata('keywords',
↳$this->category->metakey);
    }
    elseif (!$this->category->metakey && $this->params->get(
↳'menu-meta_keywords')) {
      $this->document->setMetadata('keywords',
↳$this->params->get('menu-meta_keywords'));
    }

    if ($this->params->get('robots')) {
      $this->document->setMetadata('robots',
↳$this- >params->get('robots'));
    }

    if ($app->getCfg('MetaTitle') == '1') {
      $this->document->setMetaData('title',
↳$this->category->getMetadata()->get('page_title'));
    }

    if ($app->getCfg('MetaAuthor') == '1') {
      $this->document->setMetaData('author',
↳$this->category->getMetadata()->get('author'));
    }

    $mdata = $this->category->getMetadata()->toArray();

    foreach ($mdata as $k => $v)
    {
      if ($v) {
        $this->document->setMetadata($k, $v);
      }
    }
  }
} // конец класса
```

В этом фрагменте кода в документ вводятся метаданные из категории или пункта меню. И в этом случае используются как можно более конкретные метаданные из тех, что доступны.

Модель

В предыдущем разделе был рассмотрен исходный код, в котором из модели вызывались методы `getState()`, `getItems()`, `getCategory()` и `getPagination()`. Эта модель определяется в классе `JoomproSubsModelCategory` из файла `models/category.php`. Рассмотрим его исходный код, первая часть которого приведена ниже.

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

jimport('joomla.application.component.modellist');
jimport('joomla.application.categories');
/**
 * Модель для компонента Joomprosubs
 *
 * @package       Joomla.Site
 * @subpackage    com_joomprosubs
 */
class JoomprosubsModelCategory extends JModelList
{
  /**
   * Данные для элементов категории
   *
   * @var   array
  */
  protected $_item = null;

  /**
   * Конструктор.
   *
   * @param       array       Дополнительный ассоциативный массив
   *                          с параметрами настройки модели.
   * @see         JController
  */
  public function __construct($config = array())
  {
    if (empty($config['filter_fields'])) {
        $config['filter_fields'] = array(
          'id', 'a.id',
          'title', 'a.title',
          'g.title', 'group_title',
          'duration', 'a.duration'
        );
    }

    parent::__construct($config);
  }
```

Рассматриваемый здесь класс модели расширяет класс `JModelList`, который, в свою очередь, расширяет класс `JModel`. Следует иметь в виду, что метод `getState()`

наследуется из класса JModel, тогда как метод getPagination() — из класса JModelList. Кроме того, в классе модели импортируется класс JCategories, используемый в методе getCategory(). В нем также определяется поле $_item, используемое для хранения объекта категории. В конструкторе данного класса создается массив полей фильтров, предназначенных для проверки достоверности данных упорядочения столбцов. После этого вызывается конструктор родительского класса.

Далее в этом классе определяется метод getListQuery(). Ниже приведена первая часть исходного кода этого метода.

```
protected function getListQuery()
{
  $user = JFactory::getUser();
  $groups = implode(',', $user->getAuthorisedViewLevels());

  // создать новый объект запроса
  $db          = $this->getDbo();
  $query  = $db->getQuery(true);

  // выбрать требуемые поля из категорий
  $query->select($this->getState('list.select', 'a.*'));
  $query->select('g.title as group_title');
  $query->from($db->quoteName('#__joompro_subscriptions').' AS a');

  // осуществить соединение по группам, чтобы получить название группы
  $query->join('LEFT', $db->quoteName('#__usergroups').' AS g ON
↳a.group_id = g.id');
  $query->where('a.access IN ('.$groups.')');

  // отфильтровать по категории
  if ($categoryId = $this->getState('category.id')) {
    $query->where('a.catid = '.(int) $categoryId);
    $query->join('LEFT', $db->quoteName('#__categories').' AS c ON
↳c.id = a.catid');
    $query->where('c.access IN ('.$groups.')');

  // отфильтровать по опубликованной категории
  $cpublished = $this->getState('filter.c.published');
  if (is_numeric($cpublished)) {
    $query->where('c.published = '.(int) $cpublished);
  }
}
```

В данном методе формируется запрос на компоновку списка. В этот запрос включаются все столбцы из таблицы #__joompro_subscriptions и заголовок из таблицы групп пользователей. Подписки включаются только в ту группу, к которой пользователю разрешен доступ, а затем они фильтруются по идентификатору категории и ее состоянию "опубликовано".

Ниже приведена последняя часть исходного кода из метода getListQuery().

```
// отфильтровать по состоянию
$state = $this->getState('filter.state');
if (is_numeric($state)) {
```

```
    $query->where('a.published = '.(int) $state);
  }

  // отфильтровать по критериям поиска
  if ($this->getState('list.filter') != '') {
    $filter = JString::strtolower($this->getState('list.filter'));
    $filter = $db->quote('%'.$filter.'%', true);
    $query->where('a.title LIKE ' . $filter);
  }

  // отфильтровать по начальной и конечной датам
  $nullDate = $db->quote($db->getNullDate());
  $nowDate = $db->quote(JFactory::getDate()->toSQL());
  if ($this->getState('filter.publish_date')){
    $query->where('(a.publish_up = ' . $nullDate . '
↳OR a.publish_up <= ' . $nowDate . ')');
    $query->where('(a.publish_down = ' . $nullDate . '
↳OR a.publish_down >= ' . $nowDate . ')');
  }

  // добавить оператор упорядочения списка
  $query->order($db->getEscaped($this->getState('list.ordering',
↳'a.title')).
    '.$db->getEscaped($this->getState('list.direction', 'ASC')));
  return $query;
}
```

В этом фрагменте кода в запрос вводятся операторы WHERE, чтобы отфильтровать подписки по состоянию категории "опубликовано", любому тексту, вводимому пользователем для фильтрации, а также по начальной и конечной датам публикации подписки. И наконец, получаемый в итоге список упорядочивается по тому столбцу, на котором пользователь щелкнул кнопкой мыши, а в конечном итоге возвращается объект запроса.

Фильтр списка представляет собой текст, вводимый пользователем для фильтрации подписок по списку. В частности, пользователь может ввести текст **"Chevy"** ("Шевроле"), чтобы отобразить только те подписки, в названиях которых присутствует введенный текст. Этот текст вводится в запрос SQL, что требует особого внимания, чтобы отсеять любой злонамеренный код, вносимый в данный запрос. И делается это в методе `quote()`. Передавая этому методу логическое значение `true` в качестве второго аргумента, мы предписываем ему выполнить метод `escape()` по первому аргументу. В итоге будут экранированы любые специальные символы, которые могли быть использованы для внесения злонамеренного кода в запрос SQL.

Далее следует метод `populateState()`, исходный код которого приведен ниже.

```
protected function populateState($ordering = null, $direction = null)
{
  // инициализировать переменные
  $app = JFactory::getApplication();
  $params = JComponentHelper::getParams('com_joomprosubs');

  // представить сведения о состоянии списков
  $limit = $app->getUserStateFromRequest('global.list.limit',
```

```
↳'limit', $app->getCfg('list_limit'));
  $this->setState('list.limit', $limit);

  $limitstart = JRequest::getVar('limitstart', 0, '', 'int');
  $this->setState('list.start', $limitstart);

  $orderCol = JRequest::getCmd('filter_order', 'title');
  if (!in_array($orderCol, $this->filter_fields)) {
    $orderCol = 'ordering';
  }
  $this->setState('list.ordering', $orderCol);

  $listOrder = Request::getCmd('filter_order_Dir', 'ASC');
  if (!in_array(strtoupper($listOrder), array('ASC', 'DESC', ''))) {
    $listOrder = 'ASC';
  }
  $this->setState('list.direction', $listOrder);

  $this->setState('list.filter', JRequest::getString(
↳'filter-search'));
  $$id = JRequest::getInt('id', 0);
  $this->setState('category.id', $id);

  $user = JFactory::getUser();
  if ((!$user->authorise('core.edit.state', 'com_joomprosubs')) &&
↳(!$user->authorise('core.edit', 'com_joomprosubs'))){
      // ограничить опубликованными подписками для тех, кому не
      // разрешено редактировать подписки или их состояние
      $this->setState('filter.state', 1);

     // отфильтровать по начальной и конечной датам
     $this->setState('filter.publish_date', true);
  }
}
```

В этом методе устанавливаются поля объекта состояния на основании переменной запроса. Для получения ограничения, накладываемого на количество элементов, отображаемых списком на странице, вызывается метод `getUserStateFromRequest()`. Соответствующее значение считывается из объекта текущего сеанса работы. Используя этот метод, можно сохранить значение в текущем сеансе работы и тем самым запомнить то, что пользователь указал для данного экрана.

Далее получается начало накладываемого ограничения, т.е. то место в базе данных, откуда следует начинать листание длинного списка. Для того чтобы гарантировать получение целочисленного значения, вызывается метод `getInt()` из класса `JRequest`.

Затем получаются данные упорядочения столбцов и его направления. Следует иметь в виду, что для проверки достоверности данных упорядочения столбцов используется поле `filter_fields`, созданное в конструкторе класса модели. А достоверность данных о направлении упорядочения проверяется по критерию `ASC` (По восходящей) или `DESC` (По нисходящей).

После этого устанавливается поле `list.filter`, исходя из данных, введенных пользователем для фильтрации списка. Это место потенциальной уязвимости прикладного кода.

Ведь пользователь может ввести в данном поле любую символьную строку, а в методе `getString()` проверяется лишь, действительно ли введенная строка является символьной, но не отсеивается злонамеренный код. Именно поэтому при формировании запроса вызывается метод `$db->quote()`.

Далее из запроса извлекается идентификатор категории, которая устанавливается в соответствующее состояние. И наконец, если пользователю не разрешено редактировать состояние, он может лишь просматривать опубликованные элементы подписки.

И последним в рассматриваемом здесь файле следует метод `getCategory()`. Ниже приведен его исходный код.

```
  public function getCategory()
  {
    if(!is_object($this->_item))
    {
      $categories = JCategories::getInstance('Joomprosubs');
      $this->_item = $categories->get($this->getState('category.id',
↳'root'));
    }

    return $this->_item;
  }
} // конец класса
```

В этом методе объект категории просто получается по ее идентификатору. Следует, однако, иметь в виду, что это делается лишь один раз, а результат сохраняется в поле `$_item`. Если этот объект потребуется еще раз, его достаточно извлечь из данного поля.

Вспомогательный файл категорий

При вызове метода `getInstance()` из класса `JCategories`, в свою очередь, вызывается метод-конструктор `__construct()` из класса `JoomproSubsCategories`, объявляемого во вспомогательном файле `helpers/category.php`. Ниже приведен исходный код этого класса.

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

// Вспомогательный файл компонента
jimport('joomla.application.component.helper');
jimport('joomla.application.categories');

/**
 * Дерево категорий компонента Joomprosubs
 *
 * @static
 * @package       Joomla.Site
 * @subpackage    com_joomprosubs
 */
class JoomproSubsCategories extends JCategories
{
  public function __construct($options = array())
  {
    $options['table'] = '#__joompro_subscriptions';
```

```
    $options['extension'] = 'com_joomprosubs';
    $options['statefield'] = 'published';
    parent::__construct($options);
  }
} // конец класса
```

В этом вспомогательном файле предоставляются сведения о категориях, необходимые для данного расширения, в том числе имя таблицы, имя расширения, а также наличие столбца `published`, где хранится информация состояния.

Файлы компоновки категорий

Для воспроизведения списка категорий используются два файла компоновки. В качестве точки входа служит файл `views/category/tmpl/default.php`. Ниже приведен исходный код из этого файла компоновки.

```
<?php
/**
 * @package      Joomla.Site
 * @subpackage   com_joomprosubs
 * @copyright    Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↳reserved.
* @license      GNU General Public License version 2 or later; see LICENSE.txt
 */
// прямой доступ отсутствует
defined('_JEXEC') or die;
JHtml::addIncludePath(JPATH_COMPONENT.'/helpers'); ?>
<div class="joomprosub-category<?php echo $this->pageclass_sfx;?>">
<?php if ($this->params->def('show_page_heading', 1)) : ?>
<h1>
  <?php echo $this->escape($this->params->get('page_heading')); ?>
</h1>
<?php endif; ?>
<?php if($this->params->get('show_category_title', 1)) : ?>
<h2>
  <?php echo JHtml::_('content.prepare', $this->category->title); ?>
</h2>
<?php endif; ?>
<?php if ($this->params->get('show_description', 1) ||
↳$this->params->def('show_description_image', 1)) : ?>
  <div class="category-desc">
  <?php if ($this->params->get('show_description_image') &&
↳$this->category->getParams()->get('image')) : ?>
      <img src="<?php echo $this->category->getParams()->get('image');
↳?>"/>
  <?php endif; ?>
  <?php if ($this->params->get('show_description') &&
↳$this->category->description) : ?>
      <?php echo JHtml::_('content.prepare',
↳$this->category->description); ?>
  <?php endif; ?>
  <div class="clr"></div>
  </div>
```

```
<?php endif; ?>
<?php echo $this->loadTemplate('items'); ?>
</div>
```

И этот код похож на исходный код из соответствующего файла, предназначенного для компоновки списка категорий веб-ссылок или других элементов. Сначала в нем отображается описание категории и ее изображение, а затем вызывается метод `loadTemplate()` для загрузки файла исходной компоновки `default_items.php`. Ниже приведена первая часть содержимого этого файла.

```
<?php
/**
 * @subpackage    com_joomprosubs
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.
txt
 */

  // прямой доступ отсутствует
  defined('_JEXEC') or die;

  // Код для поддержки ссылок на редактирование подписок.
  // создать краткую форму для описания параметров
  JHtml::addIncludePath(JPATH_COMPONENT.'/helpers/html');
  JHtml::_('behavior.tooltip');
  JHtml::core();

  // получить объект пользователя
  $user = JFactory::getUser();
  // проверить, разрешено ли пользователю вводить или редактировать подписки
  $canEdit = $user->authorise('core.edit', 'com_joomprosubs.category.' .
↳$this->category->id);

  $listOrder = $this->escape($this->state->get('list.ordering'));
  $listDirn = $this->escape($this->state->get('list.direction'));
  $listFilter = $this->state->get('list.filter');
  ?>

  <?php if (empty($this->items)) : ?>
    <p> <?php echo JText::_('COM_JOOMPROSUBS_NO_JOOMPROSUBS'); ?></p>
  <?php else : ?>

  <form action="<?php echo htmlspecialchars(JFactory::getURI()->
↳toString()); ?>"
  method="post" name="adminForm" id="adminForm">
  <fieldset class="filters">
  <legend class="hidelabeltxt"><?php echo
↳JText::_('JGLOBAL_FILTER_LABEL'); ?></legend>
  <div class="filter-search">
    <label class="filter-search-lbl" for="filter-search">
    <?php echo JText::_('COM_JOOMPROSUBS_FILTER_LABEL').' ';
↳?></label>
```

```
    <input type="text" name="filter-search" id="filter-search"
      value="<?php echo $this->escape($this->state->get('list.filter'));
?>"
      class="inputbox" onchange="document.adminForm.submit();"
      title="<?php echo JText::_('COM_CONTENT_FILTER_SEARCH_DESC'); ?>"
/>
  </div>
<div class="display-limit">
  <?php echo JText::_('JGLOBAL_DISPLAY_NUM'); ?>
  <?php echo $this->pagination->getLimitBox(); ?>
</div>
</fieldset>
```

Сначала выполняются все служебные операции, необходимые для загрузки вспомогательных файлов, а также двух видов поведения по сценариям JavaScript: всплывающих подсказок и основного поведения. Затем проверяется, разрешено ли пользователю редактировать данную категорию. Если разрешено, то категория появится в виде ссылки на подписку. В противном случае она будет показана в обычном текстовом виде.

Далее получается поле сортировки в заданном порядке и направлении, а затем текст фильтра, если таковой имеется. Напомним, что пользователь может сортировать список, щелкая кнопкой мыши на заголовках столбцов. После этого проверяется наличие каких-нибудь элементов подписки для обработки. Если таковые отсутствуют, выводится сообщение, уведомляющее об отсутствии подписок в данной категории. В противном случае компоновка элементов подписки продолжается.

Затем создается элемент разметки формы. В качестве действия указывается переход по URL обратно на ту же самую страницу. Форма требуется для сортировки по столбцу и фильтрации по введенным пользователям критериям. Для этого воспроизводится поле ввода текста фильтра и раскрывающийся список ограничений на отображение.

Далее следует приведенный ниже фрагмент кода.

```
<table class="category">
  <thead><tr>
    <th class="title">
        <?php echo JHtml::_('grid.sort', 'COM_JOOMPROSUBS_GRID_TITLE',
        'a.title', $listDirn, $listOrder); ?>
    </th>
    <th class="group">
    <?php echo JHtml::_('grid.sort', 'COM_JOOMPROSUBS_GRID_GROUP',
        'g.title', $listDirn, $listOrder); ?>
    </th>
    <th class="duration">
      <?php echo JHtml::_('grid.sort', 'COM_JOOMPROSUBS_GRID_DURATION',
          'a.duration', $listDirn, $listOrder); ?>
    </th>
</tr></thead>
```

В этом фрагменте кода воспроизводятся заголовки сортируемых столбцов.

Следующий фрагмент кода приведен ниже.

```
<tbody>
<?php foreach ($this->items as $i => $item) : ?>
  <tr class="cat-list-row<?php echo $i % 2; ?>" >
```

```
  <td class="title">
    <?php if ($canEdit) : ?>
        <a href="<?php echo JRoute::_(
↳'index.php?option=com_joomprosubs&task=subscription.edit
↳&sub_id='. $item->id.'&catid='.$item->catid); ?>">
        <?php echo $item->title; ?></a>
    <?php else: ?>
        <?php echo $item->title;?>
    <?php endif; ?>
    <?php if ($this->params->get('show_description')) : ?>
        <?php echo nl2br($item->description); ?>
    <?php endif; ?>
    </td>
    <td class="item-group">
        <?php echo $item->group_title; ?>
    </td>
    <td class="item-duration">
        <?php echo $item->duration; ?>
    </td>
    </tr>
  <?php endforeach; ?>
</tbody>
</table>
```

В этом фрагменте кода организуется циклическое обращение к элементам массива и воспроизведение каждой строки таблицы, в том числе заголовка, описания, если соответствующий параметр установлен на отображение описания, а также названия подписки и срока ее действия.

И последний фрагмент кода из рассматриваемого здесь файла компоновки приведен ниже.

```
<div class="pagination">
  <p class="counter">
  <?php echo $this->pagination->getPagesCounter(); ?>
  </p>
  <?php echo $this->pagination->getPagesLinks(); ?>
</div>
<div>
  <input type="hidden" name="filter_order"
↳value="<?php echo $listOrder; ?>" />
  <input type="hidden" name="filter_order_Dir"
↳value="<?php echo $listDirn; ?>" />
</div>
</form>
<?php endif; ?>
```

В этом фрагменте кода воспроизводятся элементы управления разбиением на страницы, закрывается форма и завершается кодовый блок условного оператора `if-else`. Пример данной компоновки приведен на рис. 10.2.

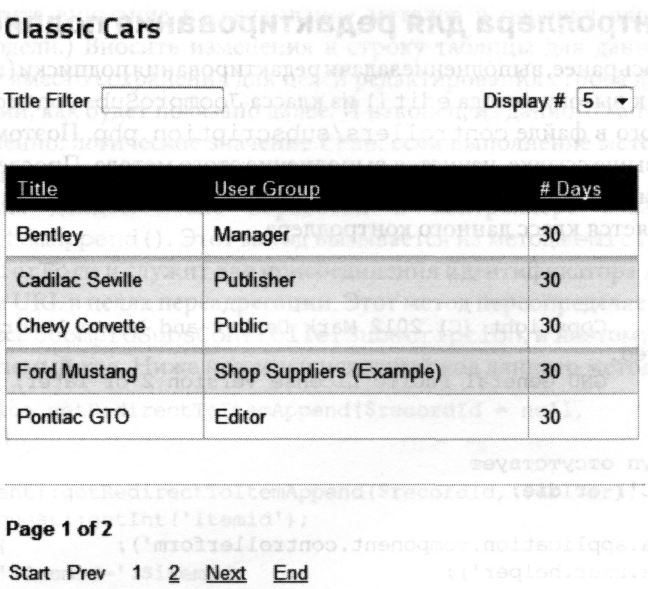


Рис. 10.2. Пример компоновки подписок

Представление подписки

А теперь рассмотрим представление подписки. Именно здесь пользователь может подписаться на одну из предлагаемых подписок. Если пользователь, которому разрешено редактировать данную категорию подписок, зарегистрирован на веб-сайте, названия подписок отображаются в виде ссылок на форму подписки. Например, подписка с идентификатором **3** отображается в виде следующей ссылки:

```
index.php/subscriptions?task=subscription.edit&sub_id=3
```

Если щелкнуть на этой ссылке, появится экран, приведенный на рис. 10.3.

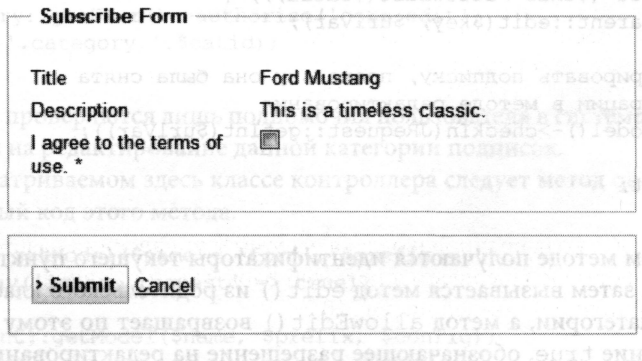


Рис. 10.3. Форма подписки

Методы контроллера для редактирования подписки

Как пояснялось ранее, выполнение задачи редактирования подписки (susbscription.edit) приводит к вызову метода edit() из класса JoomproSubsControllerSubscription, объявляемого в файле controllers/subscription.php. Поэтому если щелкнуть на упомянутой выше ссылке, начнется выполнение этого метода. Проследим логику представления подписки, начиная с контроллера. Ниже приведена первая часть исходного кода, где объявляется класс данного контроллера.

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All
↪rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

jimport('joomla.application.component.controllerform');
jimport('joomla.user.helper');

class JoomproSubsControllerSubscription extends JControllerForm
{
  protected $view_item = 'form';
```

В этом фрагменте кода импортируются все необходимые классы и создается поле $view_item, используемое далее в объявляемом классе контроллера. А далее определяется метод edit(), вызываемый после щелчка кнопкой мыши на ссылке редактируемой подписки. Код данного метода приведен ниже.

```
public function edit($key = null, $urlVar = 'sub_id')
{
  $result = false;
  $itemid = JRequest::getInt('Itemid');
  $catid = JRequest::getInt('catid');
  if (($catid) && ($this->allowEdit($catid))) {
    $result = parent::edit($key, $urlVar);

    // зарегистрировать подписку, поскольку она была снята
    // с регистрации в методе редактирования
    $this->getModel()->checkIn(JRequest::getInt($urlVar));
  }
  return $result;
}
```

Сначала в этом методе получаются идентификаторы текущего пункта меню и категории из запроса, а затем вызывается метод edit() из родительского класса, если имеется идентификатор категории, а метод allowEdit() возвращает по этому идентификатору логическое значение true, обозначающее разрешение на редактирование. Но поскольку редактируемый элемент снимается в этом методе с регистрации (т.е. блокируется на время редактирования), то он регистрируется снова (т.е. разблокируется) для использования

в модели. (Обратите внимание на связывание методов в цепочку для вызова метода checkIn() из модели.) Вносить изменения в строку таблицы для данной подписки не нужно, и поэтому вместо сохранения для целей редактирования строка добавляется в таблицу соответствий, как будет показано далее. И наконец, из данного метода возвращается результат, а именно: логическое значение true, если выполнение метода завершилось успешно, в противном случае — логическое значение false.

Следующим на данном этапе обработки в контроллере оказывается метод getRedirectToItemAppend(). Этот метод вызывается из метода edit() родительского класса JControllerForm и служит для присоединения идентификатора текущего пункта меню к запросу из URL в целях переадресации. Этот метод переопределяется в рассматриваемом здесь классе JoomproSubsControllerSubscription, и поэтому в нем выполняется переопределяемый код. Ниже приведен исходный код данного метода.

```
protected function getRedirectToItemAppend($recordId = null,
↳$urlVar = null)
{
  $append = parent::getRedirectToItemAppend($recordId, $urlVar);
  $itemId = JRequest::getInt('Itemid');
  if ($itemId) {
    $append .= '&Itemid='.$itemId;
  }
  return $append;
}
```

Сначала в этом методе вызывается одноименный метод из родительского класса, а затем идентификатор текущего пункта меню (Itemid) присоединяется к запросу из URL и, наконец, возвращается результат. Это означает, что тот же самый идентификатор текущего пункта меню сохраняется в переадресации, а следовательно, все модули и шаблоны, назначенные для данного пункта меню, будут отображаться и после переадресации к форме подписки.

Ниже приведен исходный код метода allowEdit() из рассматриваемого здесь класса контроллера.

```
protected function allowEdit($catid)
{
  return JFactory::getUser()->authorise('core.edit',
↳$this->option.'.category.'.$catid);
}
```

В этом методе проверяются лишь полномочия пользователя в системе списков контроля доступа (ACL) на редактирование данной категории подписок.

Далее в рассматриваемом здесь классе контроллера следует метод getModel(). Ниже приведен исходный код этого метода.

```
public function getModel($name = 'form', $prefix = '',
↳$config = array('ignore_request' => true))
{
  $model = parent::getModel($name, $prefix, $config);
  return $model;
}
```

В этом методе лишь указывается имя `form` используемой модели. Это означает, что полное имя модели будет следующим: `JoomproSubsModelForm`.

Представление и форма редактирования

Далее в рассматриваемом здесь процессе редактирования подписок наступает стадия, на которой выполняется метод `display()` из класса представления `JoomproSubsViewForm`, объявляемого в файле `views/form/view.html.php`. Ниже приведена первая часть исходного кода из этого файла.

```
<?php
/**
 * @package       Joomla.Site
 * @subpackage    com_joomprosubs
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↳reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;

jimport('joomla.application.component.view');

/**
 * Класс HTML-представления для компонента JoomproSubs
 *
 */
class JoomprosubsViewForm extends JView{
  protected $state;
  protected $item;
  protected $category;

  function display($tpl = null)
  {
    $app = JFactory::getApplication();
    $params = $app->getParams();
    $user = JFactory::getUser();

    // получить некоторые данные из модели
    $item = $this->get('Item');
    $this->form = $this->get('Form');
```

В этом фрагменте кода объявляются поля данного класса и начальная часть метода `display()`. Сначала в этом методе выполняются стандартные служебные операции, а затем из модели получаются элемент и форма. Эта модель определяется в классе `JoomproSubsModelForm`, расширяя модель из класса `JoomproSubsModelSubscription`, относящегося к административной части рассматриваемого здесь компонента. В классе модели из административной части имеется метод `getForm()`, наследующий методы `getState()` и `getItem()`. Когда этот метод вызывается из класса в пользовательской части, метод `getForm()` загружает файл `subscription.xml` из папки `components/`

com_joomprosubs/models/forms, но не из папки administrator/components. Дело в том, что для указания пути к данному компоненту используется константа JPATH_COMPONENT, которая имеет разное значение в пользовательской и административной частях веб-сайта.

Ниже приведено содержимое XML-файла models/forms/subscription.xml для разметки формы редактирования.

```
<?xml version="1.0" encoding="UTF-8"?>
<form>
  <fields>
  <fieldset>
    <field name="subscription_terms" type="checkbox"
        default="0" filter="bool"
        label="COM_JOOMPROSUBS_FORM_TERMS_LABEL"
        desc="COM_JOOMPROSUBS_FORM_TERMS_DESC"
        required="true"
        value="1"
    />
  </fieldset>
  </fields>
</form>
```

В этом файле размечается одно поле формы, которое представляет собой флажок subscription_terms, определяющий условия подписки.

Вернемся, однако, к классу представления JoomproSubsViewForm из файла views/form/view.html.php. Ниже приведена остальная часть исходного кода из метода display().

```
  $authorised = $user->authorise('core.edit', 'com_joomprosubs.category.'
↳. $item->catid);

  if ($authorised !== true) {
    JError::raiseError(403, JText::_('JERROR_ALERTNOAUTHOR'));
    return false;
  }

  $this->form->bind($item);

  // проверить ошибки
  if (count($errors = $this->get('Errors'))) {
    JError::raiseWarning(500, implode("\n", $errors));
    return false;
  }

  // экранировать символьные строки для вывода в формате HTML
  $this->pageclass_sfx = htmlspecialchars(
↳$params->get('pageclass_sfx'));

  $this->params = $params;
  $this->user = $user;
  $this->item = $item;
```

```
    $this->_prepareDocument();
    parent::display($tpl);
}
```

В приведенном выше фрагменте кода проверяются полномочия пользователя на редактирование категории, хотя в данном случае это не строго обязательно. Ведь мы уже проверили эти полномочия в методе `edit()` из класса рассматриваемого здесь контроллера. Тем не менее вводить такую проверку рекомендуется и в представление. Это дает возможность лишний раз убедиться в том, что у пользователя имеются подходящие полномочия, если данный метод используется в другом контексте.

Далее вызывается метод `bind()` и проверяются ошибки. Затем экранируется добавляемый суффикс класса страницы, чтобы исключить внесение любого злонамеренного кода. После этого данные сохраняются в полях `params`, `user` и `item`, чтобы ими можно было воспользоваться в файлах компоновки. И наконец, вызывается метод `_prepareDocument()` и далее метод отображения из родительского класса для обращения к файлам компоновки.

Метод `_prepareDocument()` также относится к рассматриваемому здесь классу. Ниже приведен его исходный код.

```
    protected function _prepareDocument()
    {
        $app = JFactory::getApplication();
        $menus = $app->getMenu();
        $pathway = $app->getPathway();
        $title = null;

        // приложение устанавливает заглавие страницы по умолчанию,
        // и поэтому его необходимо получить из самого пункта меню
        $menu = $menus->getActive();

        $head = JText::_('COM_JOOMPROSUBS_FORM_SUBMIT_SUB');

        if ($menu) {
            $this->params->def('page_heading',
↳$this->params->get('page_title', $menu->title));
        } else {
            $this->params->def('page_heading', $head);
        }

        $title = $this->params->def('page_title', $head);
        if ($app->getCfg('sitename_pagetitles', 0)) {
            $title = JText::sprintf('JPAGETITLE', $app->getCfg('sitename'),
↳$title);
        }
        $this->document->setTitle($title);
        if ($this->params->get('menu-meta_description'))
        {
            $this->document->setDescription($this->params->get(
↳'menu-meta_description'));
        }
```

```
    if ($this->params->get('menu-meta_keywords'))
    {
      $this->document->setMetadata('keywords',
↳$this->params->get('menu-meta_keywords'));
    }

    if ($this->params->get('robots'))
    {
      $this->document->setMetadata('robots',
↳$this->params->get('robots'));
    }
  }
} // конец класса
```

В этом методе устанавливаются заголовок документа на странице, заглавие страницы, описание, а также ключевые слова и роботы. Этот метод мало чем отличается от аналогичных методов из других представлений. Но в то же время данное обстоятельство явно указывает на необходимость исключить дублирование кода, если такой же метод создан и вызывается из родительского класса.

Компоновка формы редактирования

Компоновка формы редактирования подписки находится в файле `views/tmpl/edit.php`. Ниже приведен исходный код этой компоновки.

```
<?php
  /**
   * @copyright   Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↳reserved.
   * @license     GNU General Public License version 2 or later; see LICENSE.txt
   */
  defined('_JEXEC') or die;
  JHtml::_('behavior.keepalive');
  JHtml::_('behavior.formvalidation');
  JHtml::_('behavior.tooltip');
  $itemid = JRequest::getInt('Itemid');
  ?>

<div class="edit joomprosubs-edit<?php echo $this->pageclass_sfx; ?>">
  <form action="<?php echo JRoute::_('index.php'); ?>"
      id="adminForm" name="adminForm" method="post" class="form-validate">
    <fieldset>
    <legend><?php echo JText::_('COM_JOOMPROSUBS_FORM_LABEL'); ?></legend>
      <dl>
        <dt><?php echo JText::_('COM_JOOMPROSUBS_GRID_TITLE'); ?></dt>
        <dd><?php echo $this->escape($this->item->get('title')); ?></dd>
        <dt><?php echo JText::_('COM_JOOMPROSUBS_GRID_DESC'); ?></dt>
        <dd><?php echo $this->escape(strip_tags($this->
↳item->get('description'))); ?></dd>
        <dt><?php echo $this->form->getLabel('subscription_terms');
↳?></dt>
        <dd><?php echo $this->form->getInput('subscription_terms');
↳?></dd>
```

```
      </dl>
    </fieldset>
    <fieldset>
      <button class="button validate" type="submit"><?php echo
↳JText::_('COM_JOOMPRPOSUBS_FORM_SUBMIT'); ?></button>
      <a href="<?php echo
↳JRoute::_('index.php?option=com_joomprosubs&Itemid='. $itemid); ?>">
        <?php echo JText::_('COM_JOOMPRPOSUBS_FORM_CANCEL'); ?></a>
      <input type="hidden" name="option" value="com_joomprosubs" />
      <input type="hidden" name="task" value="subscription.subscribe" />
      <input type="hidden" name="sub_id" value="<?php echo
↳$this->item->id; ?>" />
      <?php echo JHtml::_( 'form.token'); ?>
    </fieldset>
  </form>
</div>
```

В этой компоновке сначала создается форма, а затем одно ее поле воспроизводится с помощью методов `getLabel()` и `getInput()`. После этого создается кнопка предъявления данной формы, ставящая задачу, равнозначную подписанию на конкретную подписку (`subscription.subscribe`). Напомним, что в файле `subscription.xml` атрибуту `required` присваивается логическое значение `true`, требующее установки соответствующего флажка. Если пользователь щелкнет на кнопке предъявления формы, не установив этот флажок, то получит сообщение об ошибке "`Field required: I agree to the terms of use`" (Требуется заполнить поле "Соглашаюсь с условиями использования"). Следует также иметь в виду, что в данной форме предоставляется ссылка на отмену для возврата к исходному пункту меню.

Задача подписки

Если пользователь щелкнет на кнопке предъявления формы, в рассматриваемом здесь контроллере будет выполнен метод `subscribe()`, определяемый в классе `JoomproSubsController Subscription` из файла `controllers/subscription.php`. Ниже приведена первая часть исходного кода этого метода.

```
/**
 * Подписывает на конкретную подписку.
 *
 * @param   string      $key            Имя первичного ключа в переменной URL.
 * @param   string      $urlVar         Имя переменной URL, отличающееся от
 *                                      имени первичного ключа.
*
* @return   string                     Возвращаемый URL.
*/
public function subscribe ($key = null, $urlVar = 'sub_id')
{
  // проверить полномочия пользователя
  $user = JFactory::getUser();
  if (!$user->authorise('core.edit', 'com_joomprosubs.category.' .
↳$this->category->id)) {
    JError::raiseError(403, JText::_('JERROR_ALERTNOAUTHOR'));
    return false;
```

```
  }

  // проверить достоверность данных в форме
  if(!$this->validate()) {
    return false;
  }

  // ввести пользователя в группу, если он еще не является ее членом
  $model = $this->getModel();
  $id = JRequest::getInt('sub_id');
  $subscription = $model- >getItem($id);

  // установить переадресацию без идентификатора на случай ошибки
  $this->setRedirect(JRoute::_(
↳'index.php?option=com_joomprosubs&view=form&layout=thankyou', false));
```

И в этом случае сначала проверяются полномочия пользователя. Ведь злоумышленник может просто изменить форму, чтобы добраться до нужной задачи и идентификатора, и поэтому необходимо убедиться в том, что данный пользователь является законным. После этого вызывается метод `validate()`, чтобы проверить достоверность данных в форме. Более подробно этот метод будет рассматриваться ниже. Затем происходит обращение к модели с целью получить элемент подписки. И наконец, устанавливается ссылка для переадресации без идентификационного номера. Как будет показано далее, эта ссылка указывает на компоновку экрана с сообщением о неудачном завершении операции.

Обратите внимание на то, что метод `JRoute::_()` вызывается с логическим значением `false` в качестве второго аргумента. По умолчанию в качестве второго аргумента этого метода указывается логическое значение `true`, что приводит к вызову функции `htmlspecialchars()` для экранирования специальных символов в URL. Напомним, что в этой функции символ **&** заменяется на **&**. Когда метод `JRoute::_()` используется для построения URL, то, как правило, второй его аргумент опускается, а следовательно, он принимает выбираемое по умолчанию значение. Но если метод `JRoute::_()` используется для построения URL в целях переадресации, то в качестве второго его аргумента следует указать логическое значение `false`, чтобы не экранировать специальные символы в URL. В противном случае при обработке переадресации переменные запроса будут указаны в URL неверно (например, вместо ключа `Itemid` в массиве `$_REQUEST` окажется `&Itemid`).

Ниже приведена остальная часть исходного кода метода `subscribe()`.

```
  if (!in_array($subscription->group_id, $user->groups)) {
    if (!JUserHelper::addUserToGroup($user->id, $subscription->group_id)) {
      $this->setMessage($model->getError(), 'error');
     return false;
    }
  }

  // добавить или обновить строку в таблице соответствий
  if (!$result = $model->updateSubscriptionMapping($subscription,
↳$user)) {
    $this->setMessage($model->getError(), 'error');
    return false;
```

```
    }

    // до сих пор все складывалось удачно,
    // инициировать событие onAfterSubscribe
    JDispatcher::getInstance()->trigger('onAfterSubscribe',
array(&$subscription));

    // включить идентификатор в переадресацию для сообщения
    // об удачном завершении операции
    $this->setRedirect(JRoute::_(
'index.php?option=com_joomprosubs&view=form&layout=thankyou
&sub_id='.$id, false));
    return true;
}
```

В этом фрагменте кода в базу данных вносятся изменения, отражающие тот факт, что пользователь совершил подписку. Напомним, что для совершения подписки требуется выполнить два действия. Во-первых, ввести пользователя в группу, сформированную в системе списков контроля доступа (ACL) для данной подписки. И во-вторых, добавить строку в таблицу соответствий, чтобы записать в ней подробные сведения о подписке.

Для этого сначала проверяется, не является ли пользователь уже членом данной группы. Если он им не является, то для ввода пользователя в группу вызывается метод `JUserHelper::addUserToGroup()`. И если эта операция завершится неудачно, то составляется сообщение об ошибке и возвращается логическое значение `false`.

В следующей строке кода из модели вызывается метод `updateSubscriptionMapping()`, и если он возвращает логическое значение `false`, то устанавливается состояние ошибки. Подробнее об этом речь пойдет несколько позже. А далее для рассматриваемого здесь компонента инициируется событие `onAfterSubscribe`, чтобы найти любые подключаемые модули с таким же именем метода и вызвать их. Для нормальной работы данного компонента это событие не существенно. Его можно инициировать по желанию. И наконец, для переадресации устанавливается полное значение маршрута, включая идентификатор. Это значение будет использовано в компоновке, чтобы проверить, насколько успешно была выполнена поставленная задача. Обратите внимание на то, что и в данном случае в качестве второго аргумента при вызове метода `JRoute::_()` указывается логическое значение `false`.

Метод контроллера для проверки достоверности

В методе `subscribe()` для проверки достоверности данных вызывался метод `validate()`. Ниже приведен исходный код этого метода.

```
/**
 * Проверяет достоверность данных
 *
 * @return      boolean     Истинно, если данные достоверны, а иначе – ложно.
 */
protected function validate()
{
  $app = JFactory::getApplication();
  $model = $this->getModel();
```

```
    $data = JRequest::getVar('jform', array(), 'post', 'array');
    $form = $model->getForm($data, false);
    $validData = $model->validate($form, $data);
    $recordId = JRequest::getInt('sub_id');

    // выявить ошибки в результате проверки достоверности данных
    if ($validData === false) {
      // получить сообщения о проверке достоверности
      $errors = $model->getErrors();

      // выдать пользователю до трех сообщений об ошибках
      for ($i = 0, $n = count($errors); $i < $n && $i < 3; $i++)
      {
        if (JError::isError($errors[$i])) {
          $app->enqueueMessage($errors[$i]->getMessage(), 'warning');
        }
        else {
          $app->enqueueMessage($errors[$i], 'warning');
        }
      }

      // сохранить данные в текущем сеансе работы
      if (isset($data[0])) {
        $app->setUserState($context.'.data', $data);
      }

      // переадресовать обратно к экрану редактирования
      $this->setRedirect(JRoute::_('index.php?option='.
↳$this->option.'&view='.$this->view_item.
↳$this->getRedirectToItemAppend($recordId, 'sub_id'), false));
      return false;
    }
    return true;
  }

} // конец класса
```

В этом методе сначала получаются данные формы из запроса, а затем из модели вызывается метод `validate()`. Напомним, что при этом выполняются методы `filter()` и `validate()` из объекта формы. Если в любом из этих методов будет обнаружена ошибка, то метод `validate()` из модели возвратит логическое значение `false`. В этом случае метод `validate()` из контроллера выведет на экран до трех сообщений об ошибках, а затем установит переадресацию обратно к форме редактирования подписки.

Модель для формы

Как пояснялось при обсуждении шаблона проектирования MVC, модель, как правило, отвечает за взаимодействие с базой данных. В данном случае из модели вызывается метод `updateSubscriptionMapping()` для обработки тех изменений в базе данных, которые требуется внести в таблицу соответствий.

Класс модели `JomproSubsModelForm` объявляется в файле `models/form.php`. Ниже приведена первая часть исходного кода этого класса модели.

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All rights
↳reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

  // прямой доступ отсутствует
  defined('_JEXEC') or die;
JLoader::register('JoomproSubsModelSubscription',
↳JPATH_COMPONENT_ADMINISTRATOR.'/models/subscription.php');

/**
 * Модель компонента Joomprosubs.
 *
 * @package       Joomla.Site
 * @subpackage    com_joomprosubs
 */
class JoomproSubsModelForm extends JoomproSubsModelSubscription
{
```

Как видите, этот класс расширяет класс модели из административной части, что позволяет вызывать методы, необходимые для получения элементов подписки. Обратите также внимание на применение метода `JLoader::register()`, которому отдается предпочтение над командой `require_once`.

В данном классе модели определяется единственный открытый метод `updateSubscriptionMapping()`. Ниже приведен исходный код этого метода.

```
/**
 *
 * Метод для ввода или обновления таблицы соответствий подписок.
 * Если строка уже имеется в таблице, обновляет начальную и конечную
 * даты подписки, в противном случае добавляет в таблицу новую строку.
 *
 * @param   Jobject     $subscription     Объект подписки.
 * @param   JUser $user                   Объект пользователя.
 * @return  boolean                       Истинно, если операция завершается
 *                                        успешно, а иначе – ложно.
 */
public function updateSubscriptionMapping($subscription, $user)
{
  // проверить достоверность введенных данных
  if (((int) $subscription->id) && ((int) $subscription->duration)
↳&& ((int) $user->id)) {
    $today = JFactory::getDate()->toSQL();
    $endDate = JFactory::getDate('+ ' . (int)
↳$subscription->duration . ' days')->toSQL();

    // проверить существование строки в таблице соответствий
    $mapRow = $this->getMapRow($subscription->id, $user->id);
    if ($mapRow === false) {
      // ошибка в базе данных
      return false;
```

```
    } else if ($mapRow) {
      // строка уже имеется в таблице соответствий, поэтому обновить ее
      if (!$this->updateMapRow($subscription->id, $user->id, $today,
↳$endDate)) {
        return false;
      }
    } else {
      // строка отсутствует в таблице соответствий,
      // поэтому добавить в нее новую строку
      if (!$this->addMapRow($subscription->id, $user->id, $today,
↳$endDate)) {
        return false;
      }
    }

    // в данный момент база данных успешно обновлена
    return true;
  }
}
```

Сначала в данном методе проверяется наличие действительных ненулевых целочисленных идентификаторов пользователя и подписки, а также срок ее действия, и затем рассчитывается конечная дата подписки с помощью метода `getDate()` из класса `JFactory`. Этот метод принимает в качестве своего аргумента промежуток времени относительно текущего момента времени, например `"+ 10 minutes"` (10 минут) или `"+ 30 days"` (30 дней). В данном случае в качестве аргумента ему передается промежуток времени `"+ xx days"`, где `xx` — срок действия подписки в днях.

Далее вызывается метод `getMapRow()`, чтобы выяснить, имеется ли искомая строка в таблице соответствий. Если такая строка имеется, то вызывается метод `updateMapRow()`, чтобы обновить поля в этой строке. В противном случае вызывается метод `addMapRow()`, чтобы добавить строку в таблицу.

Ниже приведен исходный код метода `getMapRow()`.

```
protected function getMapRow($subID, $userID)
{
  $db = $this->getDbo();
  $query = $db->getQuery(true);
  $query->select('subscription_id, user_id, start_date, end_date');
  $query->from($db- >nameQuote('#__joompro_sub_mapping'));
  $query->where('subscription_id = ' . (int) $subID);
  $query->where('user_id = ' . (int) $userID);
  $db->setQuery($query);
  $data = $db->loadObject();
  if ($db->getErrorNum()) {
    $this->setError(JText::_('COM_JOOMPROSUBS_GET_MAP_ROW_FAIL'));
    return false;
  } else {
    return $data;
  }
}
```

В этом методе выбирается запрос на получение строки из таблицы по идентификаторам подписки и пользователя, а в итоге возвращается стандартный объект. Напомним, что в таблице соответствий эти значения служат в качестве первичного ключа. Это означает, что для каждой комбинации идентификаторов подписки и пользователя в таблице имеется лишь одна строка.

Ниже приведен исходный код метода `updateMapRow()`.

```
protected function updateMapRow($subID, $userID, $startDate, $endDate)
{
  $db = $this->getDbo();
  $query = $db->getQuery(true);
  $query->update($db->nameQuote('#__joompro_sub_mapping'));
  $query->set('start_date = ' . $db->quote($startDate));
  $query->set('end_date = ' . $db->quote($endDate));
  $query->where('subscription_id = ' . (int) $subID);
  $query->where('user_id = ' . (int) $userID);
  $db->setQuery($query);
  if ($db->query()) {
    return true;
  } else {
    $this->setError(JText::_('COM_JOOMPROSUBS_UPDATE_MAP_ROW_FAIL'));
    return false;
  }
}
```

В данном методе формируется запрос на обновление начальной и конечной дат в строке таблицы. И в этом случае указываются столбцы с идентификаторами подписки и пользователя, чтобы получить доступ к нужной строке таблицы.

Далее следует метод `addMapRow()`, исходный код которого приведен ниже.

```
protected function addMapRow ($subID, $userID, $startDate, $endDate)
{
  $db = $this->getDbo();
  $query = $db->getQuery(true);
  $query->insert($db->nameQuote('#__joompro_sub_mapping'));
  $query->set('subscription_id = ' . (int) $subID);
  $query->set('user_id = ' . (int) $userID);
  $query->set('start_date = ' . $db->quote($startDate));
  $query->set('end_date = ' . $db->quote($endDate));
  $db->setQuery($query);
  if ($db->query()) {
    return true;
  } else {
    $this->setError(JText::_('COM_JOOMPROSUBS_ADD_MAP_ROW_FAIL'));
    return false;
  }
}
```

В этом методе сначала формируется запрос на ввод строки в таблицу и задаются столбцы для идентификаторов подписки и пользователя, а также начальной и конечной дат. Затем составляется сообщение об ошибке, если обработка запроса завершится

неудачно. Обратите внимание на то, что для обращения с датами используется метод quote() из класса JDatabase. В данном примере даты указываются в формате "гг-мм-дд<пробел>чч:мм:сс" (например, "2012-08-10 22:40:45"). Их приходится заключать в кавычки, поскольку они содержат символ пробела.

И последним в рассматриваемом здесь классе модели является метод populateState(). Этот метод вызывается в тот момент, когда в представлении вызывается метод get('Item'). Ниже приведен исходный код метода populateState().

```
protected function populateState()
{
  $app = JFactory::getApplication();

  // загрузить состояние из запроса
  $pk = JRequest::getInt('sub_id');
  $this->setState('joomprosub.sub_id', $pk);
  // добавить переменную совместимости с принятыми по
  // умолчанию условными обозначениями
  $this->setState('form.id', $pk);

  $return = JRequest::getVar('return', null, 'default', 'base64');

  if (!JUri::isInternal(base64_decode($return))) {
    $return = null;
  }

  $this->setState('return_page', base64_decode($return));
  // загрузить параметры
  $params = $app->getParams();
  $this->setState('params', $params);
  $this->setState('layout', JRequest::getCmd('layout'));
}
```

В этом методе устанавливаются переменные для идентификаторов подписки и формы, страницы возврата, параметров и компоновки. Следует иметь в виду, что объект параметров в настоящий момент не используется. Но к нему можно будет обратиться в дальнейшем, если потребуется ввести параметры.

Компоновка экрана с благодарностями за подписку

На последней стадии выполнения задачи подписки пользователю выдается сообщение об успешном завершении этой задачи. И делается это на компонуемом экране с благодарностями за подписку. Напомним, что в методе subscribe() из класса контроллера устанавливается переадресация по следующему URL:

```
$this->setRedirect(JRoute::_('index.php?option=com_joomprosubs
&view=form&layout=thankyou&sub_id='.$id, false));
```

Если задача подписки завершится неудачно, то из этого URL исключается идентификационный номер подписки sub_id. При выполнении этой переадресации начинается новый цикл исполнения, в котором загружается представление с файлом компоновки views/form/tmpl/thankyou.php. Ниже приведено содержимое этого файла.

```
<?php
/**
 * @package      Joomla.Site
 * @subpackage   com_joomprosubs
 * @copyright    Copyright (C) 2012 Mark Dexter and Louis Landry. All
↳rights reserved.
 * @license      GNU General Public License version 2 or later; see LICENSE.txt
 */

// прямой доступ отсутствует
defined('_JEXEC') or die;
$id = (int) $this->item->id;
$name = $this->escape(JFactory::getUser()->get('name'));
$title = $this->escape($this->item->title);
$duration = (int) $this->item->duration;
$itemid = JRequest::getInt('Itemid');
?>
<?php if ($id) :?>
    <h1><?php echo JText::sprintf('COM_JOOMPROSUBS_THANK_YOU_NAME',
↳$name)?></h1>
    <p><?php echo JText::sprintf('COM_JOOMPROSUBS_THANK_YOU_TITLE',
↳$title)?></p>
    <p><?php echo JText::sprintf('COM_JOOMPROSUBS_THANK_YOU_DURATION',
↳$duration)?></p>
<?php else : ?>
    <p><?php echo JText::sprintf('COM_JOOMPROSUBS_THANK_YOU_ERROR')?></p>
<?php endif; ?>
<br/>
<a href="<?php echo JRoute::_('index.php?option=com_joomprosubs&Itemid=' .
↳$itemid); ?>" >
<?php echo JText::_('COM_JOOMPROSUBS_RETURN_TO_LIST')?></a>
```

В этом коде компоновки на экран вместе с именем пользователя выводится сообщение, уведомляющее об удачном или неудачном завершении задачи подписки. В качестве примера на рис. 10.4 приведен экран после успешного выполнения задачи подписки.

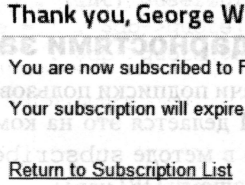


Рис. 10.4. Экран с благодарностями за успешно совершенную подписку

А на рис. 10.5 показан экран с сообщением о неудачном завершении задачи подписки. Следует, однако, иметь в виду, что полностью переведенный текст сообщений не будет отображаться на этих экранах до тех пор, пока программирование данного компонента не завершится, включая и составление файла языковой поддержки.

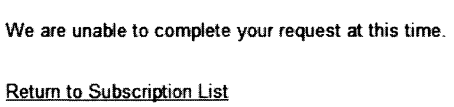


Рис. 10.5. Экран с сообщением о неудачно завершенной подписке

Как правило, выполнение задачи должно всегда завершаться успешно. Как же тогда проверить условие ее неудачного завершения? Это можно сделать несколькими способами. Самый простой из них состоит во временном изменении кода, чтобы заранее обусловить неудачное завершение метода. Например, в методе `subscribe()` из контроллера подписки имеется следующий фрагмент кода:

```
  // установить переадресацию без идентификатора на случай ошибки
  $this->setRedirect(JRoute::_(
↳'index.php?option=com_joomprosubs&view=form&layout=thankyou', false));
  if (!in_array($subscription->group_id, $user->groups)) {
    if (!JUserHelper::addUserToGroup($user->id,
↳$subscription->group_id)) {
        $this->setMessage($model->getError(), 'error');
        return false;
    }
  }
```

Если ввести в него следующую строку кода:

```
return false;
```

после вызова метода `setRedirect()`, то данный метод будет всегда возвращать логическое значение `false`, а следовательно, переход к компоновке экрана с благодарностями за подписку должен осуществляться без идентификатора подписки. (Не забудьте только удалить приведенную выше строку кода по завершении проверки!)

Файл языковой поддержки

В пользовательской части рассматриваемого здесь компонента используется лишь один файл языковой поддержки. По умолчанию это файл поддержки на британском английском языке `language/en-GB/en-GB.com_joomprosubs.ini`. И в этом случае файл языковой поддержки размещается в папке компонента, а не в общей папке языковой поддержки. Ниже приведено содержимое этого файла языковой поддержки.

```
COM_JOOMPROSUBS_ADD_MAP_ROW_FAIL="Error trying to add row to
↳mapping table."
COM_JOOMPROSUBS_DEFAULT_PAGE_TITLE="Subscriptions"
COM_JOOMPROSUBS_EDIT="Edit Subscription"
COM_JOOMPROSUBS_ERR_TABLES_NAME="There is already a Subscription with that
↳name in this category. Please try again."
COM_JOOMPROSUBS_ERR_TABLES_PROVIDE_URL="Please provide a valid URL"
COM_JOOMPROSUBS_ERR_TABLES_TITLE="Your Joomprosub must contain a title."
COM_JOOMPROSUBS_ERROR_CATEGORY_NOT_FOUND="Subscription category not found"
COM_JOOMPROSUBS_ERROR_JOOMPROSUB_NOT_FOUND="Subscription not found"
COM_JOOMPROSUBS_ERROR_JOOMPROSUB_URL_INVALID="Invalid Subscription URL"
COM_JOOMPROSUBS_ERROR_UNIQUE_ALIAS="Another Joomprosub from this category
```

```
↳has the same alias"
COM_JOOMPROSUBS_FIELD_ALIAS_DESC="The alias is for internal use only.
↳Leave this blank and Joomla! will fill in a default value from the
↳title. It has to be unique for each subscription in the same category."
COM_JOOMPROSUBS_FIELD_CATEGORY_DESC="You must select a Category."
COM_JOOMPROSUBS_FIELD_DESCRIPTION_DESC="You may enter here a description
↳for your Subscription"
COM_JOOMPROSUBS_FIELD_TITLE_DESC="Your Subscription must have a Title."
COM_JOOMPROSUBS_FILTER_LABEL="Title Filter"
COM_JOOMPROSUBS_FORM_DURATION_DESC="Number of days the subscription will
↳be active."
COM_JOOMPROSUBS_FORM_DURATION_LABEL="Duration (Days)"
COM_JOOMPROSUBS_FORM_LABEL="Subscribe Form"
COM_JOOMPROSUBS_FORM_SUBMIT_SUB="Subscribe Form"
COM_JOOMPROSUBS_FORM_TERMS_DESC="Terms for the subscription."
COM_JOOMPROSUBS_FORM_TERMS_LABEL="I agree to the terms of use."
COM_JOOMPROSUBS_FORM_TITLE_DESC="Form to subscribe to a group."
COM_JOOMPROSUBS_FORM_TITLE_LABEL="Subscribe To The Group"
COM_JOOMPROSUBS_GET_MAP_ROW_FAIL="Error trying to retrieve a row from
↳mapping table."
COM_JOOMPROSUBS_GRID_DESC="Description"
COM_JOOMPROSUBS_GRID_DURATION="# Days"
COM_JOOMPROSUBS_GRID_GROUP="User Group"
COM_JOOMPROSUBS_GRID_TITLE="Title"
COM_JOOMPROSUBS_LINK="Subscription"
COM_JOOMPROSUBS_NAME="Name"
COM_JOOMPROSUBS_NO_JOOMPROSUBS="There are no Subscriptions in this
↳category"
COM_JOOMPROSUBS_NUM="# of subscriptions:"
COM_JOOMPROSUBS_RETURN_TO_LIST="Return to Subscription List"
COM_JOOMPROSUBS_THANK_YOU_DURATION="Your subscription will expire in %s
↳days."
COM_JOOMPROSUBS_THANK_YOU_ERROR="We are unable to complete your request at
↳this time."
COM_JOOMPROSUBS_THANK_YOU_NAME="Thank you, %s."
COM_JOOMPROSUBS_THANK_YOU_TITLE="You are now subscribed to %s."
COM_JOOMPROSUBS_UPDATE_MAP_ROW_FAIL="Error trying to update a row in the
↳mapping table."
COM_JOOMPRPOSUBS_FORM_CANCEL="Cancel"
COM_JOOMPRPOSUBS_FORM_SUBMIT="Submit"
```

Обратите внимание на то, что в некоторых дескрипторах, например COM_JOOMPROSUBS_THANK_YOU_NAME, используются символы %s. Это означает, что переменная будет введена в дескриптор при переводе. В компоновке экрана с благодарностями за подписку можно обнаружить следующий код:

```
<h1><?php echo JText::sprintf('COM_JOOMPROSUBS_THANK_YOU_NAME', $name)?></h1>
```

В этом коде вместо метода _() вызывается метод sprintf() из класса JText. В качестве второго аргумента этого метода указывается переменная, добавляемая в текст, выводимый на экран (в данном случае переменная с именем пользователя). Подобная конструкция оказывается более гибкой, чем жестко программируемый порядок

следования слов в компоновке. Например, для вывода на экран можно было бы написать код, аналогичный следующему:

```
<h1><?php echo JText::_('COM_JOOMPROSUBS_THANK_YOU_NAME') . ' ' . $name)?></h1>
```

Но в таком случае имя пользователя всегда бы следовало после сообщения с благодарностями за подписку. В некоторых языках имя пользователя целесообразно указывать перед текстом сообщения или даже в середине его. И в этом отношении метод `sprintf()` предоставляет намного больше удобств при переводе выводимого на экран текста со встроенными данными.

Упаковка компонента

Итак, разработку рассматриваемого здесь компонента можно считать завершенной. Следующий, вполне логичный шаг состоит в упаковке компонента с целью упростить его установку на любом веб-сайте, совместимом с Joomla 2.5 или более поздней версией. Для этого необходимо создать архивный файл таким же образом, как это было сделано в представленных ранее примерах расширений модулей и подключаемых модулей. Ниже приводится соответствующая последовательность действий.

- Создайте рабочую папку для архивного файла (например, папку `temp`), а в ней две подпапки — `admin` и `site`.
- Скопируйте папки и файлы из папки `administrator/components/com_joomprosubs` в папку `temp/admin` на том веб-сайте на Joomla, где предполагается установить данный компонент.
- Скопируйте папки и файлы из папки `components/com_joomprosubs` в папку `temp/site`.
- Переместите файл `joomprosubs.xml` из папки `temp/admin` в папку `temp`. После этого в папке `temp` должны быть две подпапки и один XML-файл.
- Создайте архивный файл в формате zip, введя в него все содержимое папки `temp` и присвоив ему подходящее имя (например, `com_joomprosubs_1.0.0.zip`).

В итоге архивный файл должен стать устанавливаемым из компонента Extension Manager.

Новая функция: отчет о подписчиках в административной части компонента

На данном этапе разработка рассматриваемого здесь компонента в основном завершена. Но в действительности разработка любого программного обеспечения никогда не прекращается. Львиная доля задач по разработке программного обеспечения приходится на исправление программных ошибок, добавление или расширение его функциональных возможностей. Поэтому одна из главных целей шаблонов проектирования вроде MVC состоит в том, чтобы упростить расширение функциональных возможностей разрабатываемой программы, сводя к минимуму риск появления непредусмотренных побочных эффектов.

Обратимся к практическому примеру добавления новой функции в рассматриваемый здесь компонент. В данном примере добавляется новый отчет обо всех подписчиках на каждую подписку. Этот отчет будет составлен в формате файла со значениями, разделенными запятыми (CSV), пригодном для загрузки в браузер. Все подписки и подписчики на каждую из них будут представлены списком в данном отчете, а сам отчет будет доступен в качестве дополнительной функции на экране Subscription Manager.

Для того чтобы сделать этот отчет полезным, воспользуемся фильтрами компонента Subscription Manager при подготовке отчета. Иными словами, если в диспетчере подписок установлены какие-нибудь фильтры, ими можно воспользоваться для просеивания результатов составления отчета. Ниже перечислены изменения, которые требуется внести в рассматриваемый здесь компонент, чтобы реализовать эту новую функцию. Пути ко всем файлам указаны относительно папки `administrator/components/com_joomprosub`, поскольку речь идет об административной части данного компонента.

- Добавить на панели инструментов новую кнопку для загрузки файла отчета (класс `JoomproSubsViewSubmanager` из файла `views/submanager/view.html.php`).
- Дополнить контроллер новым методом для выполнения новой задачи (класс `JoomproSubsControllerSubManager` из файла `controllers/submanager.php`).
- Добавить новую модель для подготовки данных, предназначенных для отчета (класс `JoomproSubsModelCSVReport` из файла `models/csvreport.php`).
- Добавить новый метод `exportReport()`, используемый для вывода отчета вместо компоновки (класс `JoomproSubsControllerSubManager`).

Для реализации новой функции отдельная компоновка не требуется. Вместо нее будет использован метод `exportReport()` из класса `JoomproSubsControllerSubManager` для создания файла формата CSV и отправки его браузеру.

Новая кнопка панели инструментов

Напомним, что панель инструментов на экране Subscription Manager создается в методе `addToolBar()` представления из класса `JoomproSubsViewSubmanager`, объявляемого в файле `views/submanager/view.html.php`. Для ввода новой кнопки на этой панели служит приведенный ниже код.

```
// добавить кнопку экспорта на панели инструментов
$bar = JToolBar::getInstance('toolbar');
$bar->appendButton('Link', 'export', 'COM_JOOMPROSUBS_TOOLBAR_CSVREPORT',
    'index.php?option=com_joomprosubs&task=submanager.csvreport');
```

Новую кнопку можно добавить в любом месте панели инструментов, например, между пиктограммами кнопок Edit и Publish. Обратите внимание на то, что для добавления новой кнопки в приведенном выше коде используются другие методы. Так, для стандартных кнопок используются вспомогательные методы из класса `JToolBarHelper`, в том числе `editList()` и `custom()`. С их помощью кнопку на панель инструментов можно добавить в одной строке кода. А в данном случае кнопку решено добавить в виде ссылки. Это, в свою очередь, приведет к созданию нового цикла запроса на составление отчета. Таким образом, при составлении отчета в формате CSV состояние экрана диспетчера подписок не изменится. Для этого при вызове метода `appendButton()` указывается несколько аргументов.

- Тип кнопки: `'Link'` — создает кнопку типа `JButtonLink`.
- Задача: `'export'` — задает кнопку для выбора задачи экспорта на панели инструментов (служит для получения пиктограммы, обозначающей кнопку).
- Текст: `'COM_JOOMPROSUBS_TOOLBAR_CSVREPORT'` — служит для текстовой метки кнопки.
- URL: `'index.php?option=com_joomprosubs&task=submanager.csvreport'` — служит для перехода по ссылке с целью инициировать задачу составления отчета в формате CSV (`csvreport`).

Пример того, как действует кнопка типа `JButtonLink`, можно посмотреть, обратившись к исходному коду метода `back()` из класса `JToolBarHelper`.

Метод контроллера для выполнения новой задачи

После щелчка на новой кнопке панели инструментов начинается новый цикл запроса по приведенному ранее URL. Задача, указываемая в этом URL, предписывает Joomla выполнить метод **`csvReport()`** из класса `JoomproSubsControllerSubManager`, объявляемого в файле `JoomproSubsControllerSubManager`. Ниже приведен исходный код этого метода.

```
public function csvReport()
{
  $model = $this->getModel('CSVReport', 'JoomproSubsModel',
      array('ignore_request' => true));
  $model->setModelState();
  $data = $model->getItems();
  $this->exportReport($data);
}
```

Этот код краток и прост. Сначала в нем получается объект класса новой модели, а затем вызывается метод `setModelState()`. В этом методе устанавливается поле состояния модели, исходя из текущего состояния экрана диспетчера подписок. Значения состояния используются при формирования запроса. Далее из модели получаются данные. И наконец, эти данные передаются новому методу `exportReport()`.

Класс новой модели

Для новой модели придется создать отдельный класс `JoomproSubsModelCSVReport`, объявляемый в файле `models/csvreport.php`. Ниже приведена первая часть исходного кода этого класса.

```
<?php
/**
 * @copyright     Copyright (C) 2012 Mark Dexter and Louis Landry. All
↳rights reserved.
 * @license       GNU General Public License version 2 or later; see LICENSE.txt
 */

defined('_JEXEC') or die;

jimport('joomla.application.component.modellist');
JLoader::register('JoomproSubsModelSubManager',
```

```
    JPATH_COMPONENT.'/models/submanager.php');

/**
 * Методы, поддерживающие список записей подписок.
 *
 */
class JoomproSubsModelCSVReport extends JoomproSubsModelSubManager
{
  /**
   * Методы для установки состояния на основании значений,
   * выбираемых из представления подчиненного диспетчера.
   */
  public function setModelState()
  {
    $this->context = 'com_joomprosubs.submanager';
    parent::populateState();
  }
```

Как видите, этот класс расширяет класс `JoomproSubsModelSubManager`, в котором определяется модель для экрана диспетчера подписок. Это дает возможность использовать конструктор и метод `populateState()`, уже написанные для данного класса. Напомним, что в конструкторе данного класса составляется список действительных столбцов для сортировки данных. И сделано это для защиты от взлома.

Первым в рассматриваемом здесь классе объявляется метод `setModelState()`. Напомним, что отчет, составляемый в формате CSV, необходимо отфильтровать и упорядочить, используя значения, установленные на экране диспетчера подписок. Для этого необходимо вызвать метод `populateState()` со значением контекста из экрана диспетчера подписок. Но здесь имеется одно небольшое препятствие. Ведь метод `populateState()` является защищенным, а следовательно, его нельзя вызывать непосредственно из класса контроллера или переопределить (т.е. создать другой метод с таким же именем), изменив его модификатор доступа на `public` и тем самым сделав этот метод открытым. Ведь когда метод из родительского класса переопределяется в производном подклассе, у него должен быть точно такой же модификатор доступа.

Но в то же время метод `populateState()` *можно* вызвать из текущей модели, поскольку она определена в подклассе, производном от класса `JoomproSubsModelSubManager`. Следовательно, в качестве выхода из данного положения в классе новой модели можно создать (а не переопределять) *новый открытый* метод, чтобы вызвать в нем *защищенный* метод из родительского класса. Именно для того в классе `JoomproSubsModelCSVReport` и создается открытый метод `setModelState()`, чтобы воспользоваться им для вызова защищенного метода `populateState()` из родительского класса. Это даст также возможность вызывать открытый метод `setModelState()` из контроллера.

Следующим в новой модели является метод `getListQuery()`, который переопределяет аналогичный метод в родительском классе. А для этого у него должен быть такой же самый модификатор доступа (в данном случае `protected`). Ниже приведена первая часть исходного кода этого метода.

```
/**
 * Формирует запрос SQL на загрузку данных списком.
 *
 * @return JDatabaseQuery
```

```
 */
protected function getListQuery()
{
  // создать новый объект запроса
  $db = $this->getDbo();
  $query = $db->getQuery(true);

  // выбрать требующиеся поля из таблицы
  $query->select('a.id AS subsciption_id,
    a.title AS subscription_title,
    g.title AS group_title, c.title AS category_title,
    a.alias AS subscription_alias,
    a.description AS subscription_description, a.duration,
    a.published AS subscription_published,
    a.access AS subscription_access,
    uc.name AS subscriber_name');
  $query->from($db->quoteName('#__joompro_subscriptions').' AS a');
```

В этом фрагменте кода сначала создается новый объект типа `JDatabaseQuery`, а затем выбранные из таблицы столбцы вводятся в поле `select` данного объекта. Все столбцы, требующиеся для отчета, перечисляются в нужном порядке. Далее имя главной таблицы вводится в поле `from` запроса.

Ниже приведена следующая часть исходного кода метода `getListQuery()`.

```
// осуществить соединение по всей таблице соответствий,
// чтобы получить подписчиков
$query->select('m.user_id as subscriber_id, m.start_date, m.end_date');
$query->join('LEFT', $db->quoteName('#__joompro_sub_mapping').' AS m
↳ON m.subscription_id = a.id');

// осуществить соединение по всем пользователям, чтобы
// получить имя подписчика
$query->join('LEFT', $db->quoteName('#__users').' AS uc
↳ON uc.id = m.user_id');

// осуществить соединение по всем группам пользователей,
// чтобы получить имя группы
$query->join('LEFT', $db->quoteName('#__usergroups').' AS g
↳ON a.group_id = g.id');

// осуществить соединение по всем категориям
$query->join('LEFT', $db->quoteName('#__categories').' AS c ON c.id =
↳a.catid');
```

В этом фрагменте кода в запрос из таблицы соответствий вводятся столбцы с идентификатором пользователя, начальной и конечной датами. А затем осуществляется левое соединение по таблицам соответствия пользователей, групп пользователей и категорий.

Следующая часть кода рассматриваемого здесь метода приведена ниже.

```
// отфильтровать по уровню доступа
if ($access = $this->getState('filter.access')) {
    $query->where('a.access = '.(int) $access);
}
```

```
// отфильтровать по состоянию "опубликовано"
$published = $this->getState('filter.state');
if (is_numeric($published)) {
    $query->where('a.published = '.(int) $published);
} else if ($published === '') {
    $query->where('(a.published IN (0, 1))');
}

// отфильтровать по категории
$categoryId = $this->getState('filter.category_id');
if (is_numeric($categoryId)) {
    $query->where('a.catid = '.(int) $categoryId);
}

// отфильтровать по критерию поиска в названии
$search = $this->getState('filter.search');
if (!empty($search)) {
    if (stripos($search, 'id:') === 0) {
        $query->where('a.id = '.(int) substr($search, 3));
    } else {
        $search = $db->Quote('%'.$db->getEscaped($search, true).'%');
        $query->where('(a.title LIKE '.$search.'
↳OR a.alias LIKE '.$search.')');
    }
}
```

В этом фрагменте кода используются поля фильтров, выбираемые из поля состояния на экране диспетчера подписок, чтобы просеять результаты составления отчета. Для фильтрации по уровню доступа, состоянию "опубликовано", идентификатору категории и текстовому критерию поиска в запрос вводятся операторы `WHERE`. Обратите внимание на несколько усложненный характер обработки при фильтрации по текстовому критерию поиска. Если три первых символа в тексте равны `"id:"`, то поиск осуществляется по целочисленному идентификатору. Так, если ввести критерий `"id:6"` в текстовом поле поиска, то в таблице будут обнаружены только те строки, где идентификатор подписки равен **6**. В противном случае осуществляется поиск на совпадение метасимвола SQL с символами в столбце названия подписки.

И последняя часть исходного кода рассматриваемого здесь метода приведена ниже.

```
  $orderCol = $this->state->get('list.ordering', 'a.title');
  $orderDirn = $this->state->get('list.direction', 'ASC');
  $query->order($db->getEscaped($orderCol.' '.$orderDirn));

  return $query;
  }
} // конец класса
```

В этом фрагменте кода поля упорядочиваются в запросе. По умолчанию такое упорядочение происходит по названию подписки.

Метод контроллера для создания файла экспорта

Итак, у нас имеются все данные для файла экспорта. Поэтому осталось лишь создать этот файл и отправить его браузеру. И делается это в методе `exportReport()` из класса контроллера `JoomproSubsControllerSubManager`, объявляемого в файле `controllers/submanager.php`. Ниже приведен исходный код этого метода.

```
  protected function exportReport($data)
  {
    // задать заголовки
    header('Content-Type: text/csv');
    header(
↳'Content-Disposition: attachment;filename='.'subscriptions.csv');

    if ($fp = fopen('php://output', 'w')) {

      // вывести первую строку с заголовками столбцов
      if ($data[0]) {
          fputcsv($fp, array_keys(JArrayHelper::fromObject($data[0])));
      }

      // вывести строки таблицы
      foreach ($data as $row) {
          fputcsv($fp, JArrayHelper::fromObject($row));
      }

      // закрыть файл
      fclose($fp);
    }
    JFactory::getApplication()->close();
  }
} // конец класса
```

В первых двух строках кода данного метода используется функция `header()` языка PHP для отправки необработанного HTTP-заголовка браузеру, т.е. в таком формате, как показано выше. В первой строке кода браузер уведомляется о том, что ему посылается текстовый файл формата CSV, а во второй строке — о том, что файл `subscriptions.csv` посылается как вложение. В зависимости от настроек браузера пользователь может открыть или сохранить этот файл в формате CSV.

Затем в данном методе используется функция `fopen()` языка PHP для открытия объекта файла, в который можно записывать данные. Эта функция принимает два аргумента. В качестве первого из них указывается имя файла. В данном случае указано специальное имя `php://output`, позволяющее выводить результат в буфер. Оно выбрано потому, что имя файла уже указано в HTTP-заголовке. А в качестве второго аргумента данной функции указывается режим вывода данных. В данном случае указан режим `"w"`, обозначающий поток только для записи данных.

Данная функция вызывается в условном выражении оператора `if`, чтобы не производить никаких других операций с файлом, если по какой-нибудь причине выполнение функции `fopen()` завершится неудачно. В кодовом блоке условного оператора `if` проверяется наличие каких-нибудь данных в первом элементе массива. Если такие данные

имеются, то вызывается функция `fputcsv()` языка PHP для записи ключей массива в виде заголовков столбцов отчета в файле формата CSV. Функция `fputcsv()` делает все необходимое для преобразования исходных данных в текст формата CSV, для чего используется объект `$fp`, созданный ранее в функции `fopen()`.

Обратите внимание на то, что для преобразования объекта в массив вызывается метод `fromObject()` из класса `JArrayHelper`. А затем для получения ключей из массива вызывается функция `array_keys()` языка PHP. Таким образом, составляется список всех имен столбцов, размещаемых в первой строке из файла отчета.

Как только будет сформирована первая строка отчета, выполняется цикл `foreach`, в котором происходит поочередное обращение ко всем элементам массива и запись строки в файл отчета по каждому элементу. И в этом случае вызывается метод `fromObject()` для преобразования объекта в массив, прежде чем передавать его функции `fputcsv()`, а в качестве первого аргумента ей передается все тот же объект `$fp`.

И наконец, метод `fclose()` вызывается для закрытия файлового буфера. Затем вызывается функция `close()` языка PHP для объекта приложения Joomla (`$fp`). На этом цикл запроса завершается, и файл отчета отправляется браузеру.

Составление отчета

Если щелкнуть на кнопке Report (Отчет), доступной на панели инструментов диспетчера подписок, то появится диалоговое окно, аналогичное приведенному на рис. 10.6. Если же сохранить файл отчета, а затем открыть его (например, в программе электронных таблиц), то все подписки будут перечислены в формате, аналогичном показанному на рис. 10.7 и 10.8.

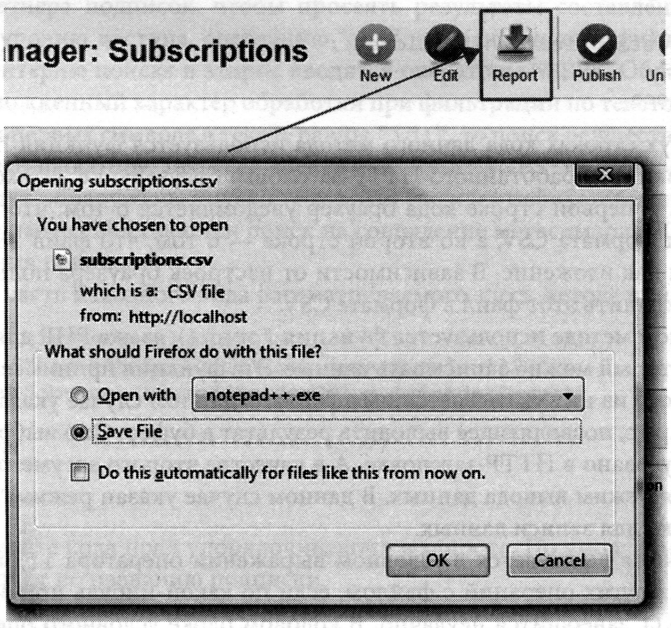


Рис. 10.6. Новая кнопка составления отчета на панели инструментов и диалоговое окно загрузки файла отчета

A	B	C	D	
subsciption_id	subscription_title	group_title	category_title	su
3	Bentley	Manager	Classic Cars	be
4	Cadilac Seville	Publisher	Classic Cars	ca
5	Chevy Corvette	Public	Classic Cars	ch
2	Ford Mustang	Shop Suppliers (Example)	Classic Cars	for
2	Ford Mustang	Shop Suppliers (Example)	Classic Cars	for
1	Pontiac GTO	Editor	Classic Cars	po

Рис. 10.7. Пример файла отчета в формате CSV. Часть I

J	K	L	M
subscriber_name	subscriber_id	start_date	end_date
George Washington	43	2011-08-11 17:12:48	2011-09-10 17:
Super User	42	2011-08-11 17:13:46	2011-09-10 17:
George Washington	43	2011-08-11 17:12:41	2011-09-10 17:

Рис. 10.8. Пример файла отчета в формате CSV. Часть II

В данном примере продемонстрирована лишь часть полезных свойств шаблона проектирования MVC. Добавить новую функцию в существующий компонент оказалось не так уж и трудно. Более того, для этого вообще не пришлось изменять уже имеющийся код, что сводит к минимуму внесение побочных эффектов, нарушающих уже действующие функции компонента.

Практические соображения

Рассмотренный здесь практический пример наглядно демонстрирует большинство аспектов разработки реального компонента. Но ради сокращения объема кода некоторые функции, которые могут потребоваться в реальном компоненте, в данном примере были опущены, в том числе следующие:

- возможность для пользователей отказаться от подписки в пользовательской части компонента;
- возможность для администраторов добавлять и удалять пользователей из подписки в административной части компонента;
- возможность для пользователей просматривать имеющиеся у них подписки и сроки их истечения.

Самый простой способ реализовать первую из перечисленных выше функций — изменить компоновку списка категорий, чтобы указать, подписался ли пользователь, и если он подписался, то когда истечет срок его подписки. Для этого потребуется также внести изменения в модель, чтобы включить в нее таблицу соответствий, а также добавить ряд полей в компоновку. Кроме того, можно было бы добавить новую кнопку для отказа от подписки, для чего потребуется и новая задача.

Для реализации второй функции можно было бы добавить новый экран диспетчера таблиц соответствий, аналогичный экрану диспетчера подписок. А третью функцию

можно было бы реализовать с помощью простого модуля, отображающего любые подписки, имеющиеся у пользователя, наряду со сроками их истечения.

Реализация любой или всех этих функций послужит вам полезным упражнением для применения на практике знаний и навыков, приобретенных в результате проработки материала четырех последних глав книги.

Резюме

В этой главе рассмотрение примера компонента подписок было завершено разработкой его пользовательской части. Это было сделано по шаблону проектирования MVC и на основании знаний, приобретенных из подробного рассмотрения компонента Weblinks. А в завершение главы разработанный компонент подписок был дополнен новой функцией составления отчета о подписках. На примере этой функции было продемонстрировано, насколько просто шаблон MVC позволяет дополнять существующие компоненты новыми функциональными возможностями и видоизменять их.

11

Работа с базой данных

В этой главе рассматриваются основы баз данных SQL и особенности работы с ними в Joomla. В начале этой главы дается краткий обзор баз данных SQL и порядок работы с ними вне среды Joomla с помощью специальной программы phpMyAdmin. В этом кратком обзоре будет показано, каким образом создаются таблицы базы данных, формируются и выполняются запросы SQL. Дав общее представление о действии запросов SQL вне среды Joomla, мы перейдем к вопросам их формирования и выполнения непосредственно в Joomla. В частности, на примере класса `JDatabaseQuery` будет сначала показано, как формируемые запросы взаимодействуют с поддерживаемыми базами данных, а затем как выполняются сформированные запросы и как обрабатываются их результаты в среде Joomla.

В качестве примера будет использована база данных MySQL, но общие принципы работы с ней распространяются на все остальные базы данных типа SQL. Имеется немало аспектов и особенностей работы с базами данных SQL, выходящих за рамки рассмотрения этой книги. Для дополнительного изучения базы данных MySQL и прочих баз данных SQL можно найти немало ресурсов в Интернете или в печатных изданиях. А в этой главе основное внимание будет уделено практическим основам работы с этими базами данных, и особенно в отношении разработки в Joomla.

Для работы с базами данных вне среды Joomla таблицы в этой главе обозначаются старым стандартным префиксом `jos` (например, `jos_categories`), а для работы в самой среде Joomla — текущим стандартным префиксом `#__`. Напомним, что префикс `#__` всегда употребляется для обращения к таблицам базы данных из приложения Joomla.

Если у вас уже имеется некоторый опыт работы с базами данных SQL вообще и с базой данных MySQL в частности, вы можете пропустить или бегло просмотреть первую часть этой главы и перейти к разделу "Работа с базой данных в среде Joomla".

Краткий обзор баз данных

Базы данных SQL, называемые также реляционными, служат основным средством для хранения данных в Joomla и других системах управления информационным наполнением веб-сайтов (CMS). Сокращение SQL обозначает язык программирования, используемый для взаимодействия с базами данных SQL. Главный принцип базы данных SQL — предоставить программе или пользователю возможность взаимодействовать с данными исключительно с точки зрения логики и безотносительно к тому, каким образом эти данные физически хранятся на компьютере. Ниже перечислены логические единицы хранения данных.

- **База данных.** Это совокупность таблиц. На веб-сайте, созданном на Joomla, как правило, используется одна база данных, которой присваивается конкретное имя при установке Joomla. Это избавляет от необходимости указывать имя базы данных после запуска Joomla на выполнение. Ведь Joomla автоматически устанавливает связь с нужной базой данных, извлекая ее имя и регистрационные данные из файла `configuration.php`.
- **Таблица.** Это совокупность строк. Ее можно рассматривать как электронную таблицу, состоящую из строк и столбцов. В каждой строке хранится по одному значению на каждый столбец.
- **Строка.** Это набор данных с одним значением для каждого столбца. Каждая строка состоит из одного или более столбца с конкретными данными. Ее структура в точности соответствует структуре столбцов электронной таблицы. Как правило, в строке хранятся различные фрагменты данных, называемые атрибутами одного элемента. Например, в строке могут храниться данные одной статьи, одного пользователя или одной записи из журнала регистрации.
- **Столбец.** Это набор данных с одним значением для каждой строки. В столбце хранятся данные конкретного типа (например, дата, текст или целое число). Так, в одном столбце может храниться число посещений статьи, в другом — имя пользователя, в третьем — временная метка (дата и время) каждой записи в журнале регистрации.

Как и электронные таблицы, те таблицы, которые хранятся в базе данных, можно просматривать как по горизонтали (по строкам), так и по вертикали (по столбцам). Просматривая таблицу по строкам, можно, как правило, обнаружить разнотипную информацию об одном элементе (например, название статьи, ее описание, дату создания), а просматривая ее по столбцам — однотипную информацию (в частности, название) о разных элементах (например, статьях). Структура таблицы базы данных задана жестко. В каждой строке имеется одинаковое количество столбцов. А каждый столбец всегда определяется для хранения данных только одного типа, например, даты, текста или целого числа. Следовательно, все данные в столбце должны быть однотипными.

Все взаимодействие с базой данных осуществляется с помощью команд языка SQL. Эти команды сообщают базе данных, *что* нужно делать, но предоставляют ей самой решать, *как* это делать. Например, базе данных совсем не обязательно указывать, где именно хранятся данные в файловой системе. Она сама найдет нужную таблицу с этими данными. На самом деле многие программисты, пишущие код для обращения к базам данных SQL,

понятия не имеют о физической структуре ее файлов (например, о выделении одного файла на каждую таблицу или же одного файла для всей базы данных в целом) либо о месте хранения этих файлов на сервере. Команды обращения к базам данных SQL не меняются при смене операционной системы, языка программирования или переносе базы данных в другую папку или на другой сервер.

Команды SQL разделяются на два общих типа: команды языка определения данных (DDL), используемые для создания или видоизменения структуры таблиц, а также команды языка манипулирования данными (DML), используемые для запроса, добавления, обновления или удаления данных из таблиц. В частности, команды DDL служат для создания первоначальной структуры базы данных, тогда как команды DML — для работы с конкретными данными.

Создание и видоизменение таблицы по командам DDL

Как упоминалось выше, команды DDL служат для создания и видоизменения структуры таблиц базы данных. В приложениях Joomla команды DDL, как правило, используются при установке, удалении или обновлении соответствующего расширения. Следовательно, эти команды чаще всего встречаются в сценариях установки и удаления, а не в исходном коде расширений на языке PHP.

И хотя командами DDL можно пользоваться динамически в цикле исполнения, делать это обычно не рекомендуется. Таблицы базы данных для большинства приложений должны быть созданы один раз: при установке или обновлении расширения. Именно поэтому команды DDL практически отсутствуют при обычной обработке данных.

В главе 9 были созданы два сценария, находящиеся в файлах `sql/install.mysql.utf8.sql` и `sql/uninstall.mysql.utf8.sql`, но эти сценарии предназначены специально для работы с MySQL. А сценарий установки Microsoft SQL Server должен находиться в файле `sql/install.sqlserver.utf8.sql`. Зачем же для различных баз данных SQL требуются разные сценарии? Дело в том, что синтаксис команд DDL в языке SQL немного отличается для различных баз данных SQL, в том числе MySQL, Microsoft SQL Server, Oracle и PostgreSQL, и поэтому сценарий, подходящий для одной базы данных, может оказаться непригодным для другой. В таком случае надежнее всего использовать разные сценарии для различных баз данных.

В рассматриваемом здесь примере сценария установки расширения применяется команда `CREATE TABLE`, по которой создаются две таблицы, используемые в этом расширении, тогда как в сценарии удаления — команда `DROP TABLE`, удаляющая (или *опускающая* в терминологии баз данных) таблицу из базы данных. Попутно заметим, что команды SQL принято указывать прописными буквами, хотя их разрешается вводить как строчными, так и прописными буквами.

В основном варианте распространения Joomla имеется файл сценария `installation/sql/mysql/joomla.sql`, содержащий все команды DDL, необходимые для создания полной базы данных в Joomla, включая основные расширения. Но опять же этот сценарий специально предназначен для базы данных MySQL, а для другой базы данных должен быть создан свой сценарий в отдельной папке. Следует также заметить, что в данном сценарии применяется команда `INSERT` типа DML для загрузки данных в основные

таблицы. Более подробно команда INSERT рассматривается далее в этой главе. В приложениях Joomla чаще всего применяются следующие команды DDL: CREATE TABLE, ALTER TABLE и DROP TABLE.

Команда CREATE TABLE

Команда CREATE TABLE является главной для уяснения особенностей построения базы данных. Ниже приведен синтаксис этой команды.

```
CREATE TABLE <имя таблицы> (<список столбцов> , <список индексов>)
<набор символов>
```

Рассмотрим команду создания базовой таблицы #__banner_clients по сценарию из файла installation/sql/mysql/joomla.sql. Ниже приведен код этой команды.

```
CREATE TABLE `#__banner_clients` (
  `id` INTEGER NOT NULL auto_increment,
  `name` VARCHAR(255) NOT NULL DEFAULT '',
  `contact` VARCHAR(255) NOT NULL DEFAULT '',
  `email` VARCHAR(255) NOT NULL DEFAULT '',
  `extrainfo` TEXT NOT NULL,
  `state` TINYINT(3) NOT NULL DEFAULT '0',
  `checked_out` INTEGER UNSIGNED NOT NULL DEFAULT '0',
  `checked_out_time` DATETIME NOT NULL default '0000-00-00 00:00:00',
  `metakey` TEXT NOT NULL,
  `own_prefix` TINYINT NOT NULL DEFAULT '0',
  `metakey_prefix` VARCHAR(255) NOT NULL default '',
  `purchase_type` TINYINT NOT NULL DEFAULT '-1',
  `track_clicks` TINYINT NOT NULL DEFAULT '-1',
  `track_impressions` TINYINT NOT NULL DEFAULT '-1',
  PRIMARY KEY (`id`),
  INDEX `idx_own_prefix` (`own_prefix`),
  INDEX `idx_metakey_prefix` (`metakey_prefix`)
) DEFAULT CHARSET=utf8;
```

Прежде всего следует заметить, что в имени создаваемой таблицы используется префикс #__. Напомним, что это заполнитель, заменяемый конкретным префиксом базы данных для отдельного веб-сайта на Joomla перед тем, как передавать команду базе данных.

Далее следует заметить, что имена таблиц и столбцов заключены в обратные кавычки (`). И хотя это не строго необходимо, тем не менее рекомендуется. Так, зарезервированные слова могут быть использованы в качестве имен таблиц и столбцов, при условии, что они заключены в обратные кавычки. Если же зарезервированное слово используется без обратных кавычек, это может привести к ошибкам при обращении к базе данных. В MySQL зарезервировано более 200 слов, в том числе BOTH, CHECK, CASCADE и SENSITIVE, поэтому ими можно легко воспользоваться, не отдавая себе отчета в данной особенности их указания в коде.

Кроме того, в новые версии MySQL иногда внедряются новые зарезервированные слова, а следовательно, слово, которое раньше не было зарезервированным, может в дальнейшем стать таковым совершенно неожиданно. Далее в этой главе будет показано, как пользоваться специальной программой phpMyAdmin для создания сценариев DDL, автоматически включающих обратные кавычки в свой код. Применение обратных кавычек

характерно лишь для MySQL, тогда как в других базах данных для этой цели могут быть использованы другие символы кавычек.

Первичные, внешние ключи и атрибут автоматического приращения

Первый столбец рассматриваемой здесь таблицы определяется следующим образом:

```
`id` INTEGER NOT NULL auto_increment
```

Как правило, первый столбец таблицы выделяется для идентификатора, как показано в приведенной выше строке кода, выполняя особую роль в таблице. Далее указываются тип данных INTEGER и два атрибута: NOT NULL и AUTO_INCREMENT. В частности, атрибут NOT NULL обозначает, что в данном столбце непременно должно храниться какое-нибудь значение, тогда как атрибут AUTO_INCREMENT имеет особое назначение и связан с определением PRIMARY KEY первичного ключа, следующим в сценарии непосредственно за определением последнего столбца таблицы.

Первичный ключ имеет совершенно однозначное значение для каждой строки таблицы. Если попытаться ввести в таблицу новую строку, в которой повторяется значение уже существующего первичного ключа (в данном случае — значение, такое же, как и у имеющегося уже идентификатора), то будет выдано сообщение об ошибке и операция ввода строки в таблицу завершится неудачно. Первичные ключи полезны тем, что по ним совершенно однозначно выбираются отдельные строки из таблицы. Как правило, столбец с первичным ключом не изменяется. Он содержит идентификатор отдельной строки, при условии, что она присутствует в таблице.

Первичные ключи нередко используются в качестве внешних ключей. Внешний ключ — это столбец таблицы A с таким же значением, как и у первичного ключа в таблице B. Например, в таблице #__weblinks базы данных Joomla имеется столбец catid. Он содержит внешний ключ к таблице #__categories, где столбец id содержит первичный ключ. В выборочных данных присутствует категория Joomla Specific Links с идентификатором **32**, а в таблице #__weblinks значение внешнего ключа в столбце catid совпадает со значением идентификатора в столбце id таблицы #__categories. Поэтому когда веб-ссылка назначается для категории Joomla Specific Links, в ее столбце catid устанавливается значение **32**, совпадающее с идентификатором в столбце id таблицы #__categories. Как будет показано далее в этой главе, внешние ключи служат для соединения нескольких таблиц в одном запросе базы данных.

Если значение каждого первичного ключа должно быть однозначным и не повторяющимся во всей таблице, то как правильно выбрать идентификатор, который еще не используется в другой строке? Для этой цели служит атрибут AUTO_INCREMENT. Он указывает базе данных создать автоматически однозначное значение для столбца id (как правило, следующее еще не использовавшееся целое значение). И когда мы вводим в таблицу новую строку, то нам уже не нужно беспокоиться о том, чтобы значение идентификатора было однозначным. Мы можем оставить его не установленным или же установить в нем пустое значение NULL. Следующее еще не использовавшееся целое значение идентификатора будет получено в базе данных MySQL автоматически, обеспечив тем самым его однозначность в столбце id.

Но в сценариях базы данных joomla.sql и sample_data.sql принят другой подход: значения для столбцов первичных ключей указываются *специально*. Это особый случай, когда заранее известно, что работа с базой данных начинается с пустой таблицы,

а следовательно, для столбцов первичных ключей можно указать значения специально, зная, что они окажутся однозначными. А когда база данных уже используется, тогда неизвестно, какие именно значения идентификаторов уже используются в ней, и поэтому необходимо, чтобы она сама выбирала эти значения.

Каждую таблицу принято определять с первичным ключом. Как правило, для этого подходит указанная выше схема: тип данных INTEGER, атрибуты NOT NULL и AUTO_INCREMENT и вполне определенный первичный ключ. Подробнее об этом речь пойдет далее. Глядя на определения остальных столбцов в рассматриваемом здесь примере, можно обнаружить в них типы данных VARCHAR, TEXT, TINYINT и DATETIME. Эти и другие типы данных рассматриваются далее в главе.

Индексы

После определений столбцов в рассматриваемом здесь примере команды CREATE TABLE следуют определения индексов. Первым из них определяется индекс PRIMARY KEY, обозначаемый как столбец `id`. Первичный ключ можно рассматривать как наиболее предпочтительный способ поиска отдельной строки в таблице. Зачастую для этой цели служит столбец типа INTEGER с атрибутом AUTO_INCREMENT, как пояснялось ранее.

В данном примере определяются два дополнительных индекса: `idx_own_prefix` и `idx_metakey_prefix`. Имена индексов принято начинать с префикса `idx_`, чтобы отличать их от имен столбцов или других имен, используемых в базе данных. Каждый индекс определяется как список, состоящий из одного или более столбца. С одной стороны, индексы помогают ускорить обработку запросов в базе данных, а с другой — замедляют в какой-то степени обновление таблицы. Поэтому индексы применяются лишь там, где они действительно способствуют обработке запросов. Применение индексов в таблицах базы данных — дело непростое, поэтому ниже приведены некоторые общие рекомендации по данному вопросу.

- Чем больше строк в таблице, тем большее значение приобретают индексы. В таблице, состоящей всего лишь из 10–20 строк, индексы вообще не нужны, кроме первичного ключа, который формально является индексом.
- Индексировать следует по столбцам, используемым для выбора, упорядочения или группирования. Например, в таблице #__content для этой цели служат индексы по столбцам access, checked_out, state, catid, created_by и language.
- Не следует индексировать по столбцам, которые обычно служат лишь для отображения, но не для выбора, упорядочения или группирования. Нецелесообразно также индексировать по очень длинным столбцам, например, по столбцам типа TEXT или MEDIUMTEXT.
- Применение индекса UNIQUE обеспечивает в базе данных требование однозначности столбца или группы столбцов в таблице. Это эффективный способ против дублирования данных. Например, в таблице #__menu используется однозначный индекс, благодаря которому на одном уровне иерархии меню не могут появиться два пункта (строки в таблице) с одним и тем же псевдонимом и языковой поддержкой.
- Если столбец содержит внешний ключ, используемый для соединения с другими таблицами, его индексация может оказаться полезной. Это еще одна причина иметь, например, индекс по столбцу catid.

Синтаксис определения индексов обладает рядом несколько запутанных особенностей. Так, слова INDEX и KEY могут использоваться попеременно. Например, вместо строки кода

```
INDEX `idx_own_prefix` (`own_prefix`),
```

можно воспользоваться следующей строкой кода:

```
KEY `idx_own_prefix` (`own_prefix`),
```

А если заменить слово UNIQUE словом INDEX, база данных MySQL получит команду обеспечить однозначность значений в указанных столбцах таблицы. Например, в определении таблицы `#__menu_types` имеется следующая строка кода:

```
UNIQUE `idx_menutype` (`menutype`)
```

Если же попытаться ввести в эту таблицу строку, дублирующую значение в уже имеющемся столбце `menutype`, то будет получено сообщение об ошибке в базе данных.

И последним в рассматриваемом здесь примере команды CREATE TABLE указывается набор символов (в данном случае — DEFAULT CHARSET=utf8). Для таблиц базы данных Joomla всегда выбирается набор символов в коде UTF-8, поскольку это дает возможность хранить в базе данных текст на языках с нелатинскими символами.

Типы данных

Как правило, выбор типа данных для столбца таблицы не составляет большого труда. Ниже дается краткое описание типов данных, наиболее употребительных в приложениях Joomla.

Тип данных INT/INTEGER

Обозначения INT и INTEGER фактически ссылаются на один и тот же целочисленный тип данных. Как пояснялось ранее, этот тип данных обычно выбирается для первичных ключей, где требуется однозначность идентификатора строки в таблице. Этот тип данных следует также выбирать для обозначения ключа в другой таблице. Например, столбец catid служит для ссылки на столбец id в нескольких таблицах, в том числе #__content и #__contacts. Поэтому для столбца catid выбирается тип данных INT, чтобы он совпадал с типом данных идентификатора в таблице #__categories.

Тип данных INT может быть как со знаком (для обозначения положительных и отрицательных значений), так и без знака (для обозначения только положительных значений). Если в определении столбца указывается атрибут AUTO_INCREMENT, то тип INT его данных всегда оказывается без знака. Иногда в коде сценариев базы данных можно встретить обозначение INT(10) типа данных INT или INTEGER. В этом случае число **10** обозначает, насколько крупным следует отображать столбец в запросе. Оно не имеет никакого отношения к максимальному значению, хранящемуся в данном столбце.

Тип данных TINYINT

Тип данных TINYINT похож на тип данных INT, за исключением того, что в нем допускаются числовые значения только в пределах от **-127** до **+127** (или от **0** до **255**). Этот тип данных применяется в тех столбцах, где присутствует небольшое количество действительных значений, например, состояние "опубликовано" или условия подтверждения/

отрицания. Применение целочисленных числовых значений данного типа дает два важных преимущества над текстовыми строками типа VARCHAR. Во-первых, при выборке целых значений из такого ненадежного источника, как форма или URL, их легко подвергнуть дополнительной обработке в коде PHP и, в частности, привести их к целочисленному типу с помощью оператора (int). Во-вторых, любые слова, используемые в приложениях Joomla, как правило, требуется переводить на выбранный язык. Для этого целые значения нетрудно связать с языковыми ключами, придав соответствующим значениям нейтральный смысл, переводимый на каждый язык.

Тип данных VARCHAR

Этот тип данных служит для хранения текстовых полей заданной длины вплоть до **21844** символов. Максимально допустимое количество символов указывается в виде числа в круглых скобках. Например, тип VARCHAR(20) допускает вводить в текстовом поле до **20** символов. Столбцы типа VARCHAR имеют переменную длину, используя лишь то место, которое требуется для хранения введенных данных. Так, если столбец имеет тип VARCHAR(100), но в нем введено всего семь символов, на диске выделяется место для хранения только этих семи символов.

Как пояснялось ранее, для всех таблиц базы данных в Joomla выбирается набор символов в коде UTF-8 (CHARSET=utf8). А поскольку символы в коде UTF-8 могут занимать до трех байтов на один символ, то максимальный размер столбцов типа VARCHAR составляет **21844** символа вместо **65535** символов для латинского алфавита.

Тип данных CHAR

В столбцах типа CHAR также могут храниться текстовые поля, но фиксированной длины. Поэтому в столбце типа CHAR(100) всегда используется место для хранения 100 символов. Максимальный размер поля типа CHAR составляет **255** символов. В Joomla тип данных CHAR используется редко, вместо него чаще применяется тип данных VARCHAR.

Тип данных DATETIME

Этот тип данных соответствует своему названию и служит для хранения даты и времени в формате "ГГ-ММ-ДД ЧЧ:ММ:СС". Имеется также тип данных DATE на тот случай, если требуется сохранить только дату, но не время. Впрочем, дату нетрудно извлечь из столбца типа DATETIME, воспользовавшись функцией date().

Тип данных TEXT

Этот тип данных похож на тип данных VARCHAR, за исключением того, что он всегда предусматривает сохранение до **65000** символов. В Joomla данные типа TEXT хранятся в столбцах, длина которых может быть, например, от 2000 до 10000 символов. К их числу относятся столбцы с параметрами (в формате JSON) и ключевыми словами. Если же столбец индексируется, то он, как правило, должен быть типа VARCHAR, а не TEXT.

Тип данных MEDIUMTEXT

Этот тип данных похож на тип данных TEXT, за исключением того, что он поддерживает максимальную длину столбца около 16 Мбайт. Тип данных MEDIUMTEXT выбирается

лишь в том случае, если длина текстового столбца может оказаться очень большой. В Joomla он используется, например, для хранения текста статей и описания категорий.

Тип данных DECIMAL

Если требуется отслеживать точные десятичные числовые значения, например денежные суммы, то для этой цели служит тип данных DECIMAL. Например, для хранения денежных сумм в долларах США с точностью до двух знаков после десятичной точки и не больше 9,999,999,999.99 можно выбрать тип данных DECIMAL(10,2).

Типы данных FLOAT и DOUBLE

Тип данных FLOAT служит для научных расчетов, где требуется достаточно большая точность. Этот тип данных обеспечивает точность до **23** значащих цифр, тогда как тип данных DOUBLE — до **53** значащих цифр. Типами данных FLOAT и DOUBLE *не* следует пользоваться для получения точных величин с фиксированным числом десятичных разрядов. Вместо них лучше выбрать тип данных DECIMAL. Ведь если выбрать тип данных FLOAT для хранения денежных сумм, то в результате их округления могут возникнуть мелкие погрешности, которые очень трудно уловить.

Атрибуты столбцов

При определении столбца можно указать не только тип данных, но и целый ряд атрибутов. К числу наиболее употребительных относятся следующие атрибуты: NOT NULL, AUTO_INCREMENT, DEFAULT и UNSIDNED.

Атрибут NOT NULL

При создании столбца в нем по умолчанию допускается устанавливать пустое значение. Следует, однако, иметь в виду, что в базах данных SQL пустое значение отличается от нулевого и может давать неожиданные или нежелательные результаты. Так, если *любое* из значений в столбце окажется пустым, то и общая сумма значений в этом столбце окажется пустой. Как правило, при определении столбцов рекомендуется не разрешать установку в них пустых значений. Для этой цели служит атрибут NOT NULL, указываемый после типа данных. И зачастую вместе с атрибутом NOT NULL указывается также атрибут AUTO_INCREMENT или DEFAULT.

Атрибут AUTO_INCREMENT

Как пояснялось ранее, этот атрибут обеспечивает автоматическое приращение числового значения первичного ключа (**1**, **2**, **3** и т.д.). Он используется вместе с типом данных INTEGER и атрибутом NOT NULL.

Атрибут DEFAULT

Этот атрибут определяет значение, устанавливаемое в столбце по умолчанию. Это значение используется при вводе новой строки без указания конкретного значения в данном столбце. Так, если отслеживаются посещения статьи или щелчки на баннере, то подсчет их количества следует начинать с нулевого, а не пустого значения. Для этого соответствующий столбец определяется следующим образом:

```
`clicks` INTEGER NOT NULL DEFAULT '0'
```

Следовательно, если включить указанное начальное значение в расчет, то оно будет трактоваться как нулевое, давая в итоге желаемый результат. Аналогичным образом обозначаются начальная и конечная даты публикации, где нулевое значение трактуется как начало или конец временного промежутка. В подобных случаях для создания столбцов используется следующий код:

```
`publish_up` DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
`publish_down` DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
```

Следует иметь в виду, что если установлен атрибут AUTO_INCREMENT, то атрибут DEFAULT не используется.

Атрибут UNSIDNED

Этот атрибут обычно указывается вместе с типом данных INTEGER или TINYINT, предписывая базе данных сохранять целочисленное значение без знака, т.е. положительным. Благодаря этому можно сохранить большее максимальное значение. Этот атрибут используется, например, в том случае, если имеется внешний ключ.

Команда ALTER TABLE

Команда ALTER TABLE похожа на команду CREATE TABLE, за исключением того, что она применяется для изменения структуры таблицы. В Joomla она чаще всего применяется для ввода новых столбцов в таблицу или изменения типа данных в уже существующей таблице. Например, в файле сценария базы данных installation/sql/mysql/joomla_update_16beta13.sql можно обнаружить следующий фрагмент кода, где употребляется эта команда:

```
ALTER TABLE `#__template_styles`
  CHANGE `params` `params` varchar(10240) NOT NULL DEFAULT '';

ALTER TABLE `#__menu`
  ADD COLUMN `client_id` TINYINT(4) NOT NULL DEFAULT 0 AFTER `language`;
ALTER TABLE `#__menu`
  ADD UNIQUE `idx_alias_parent_id` (`client_id`,`parent_id`,`alias`);
```

Первая команда в этом фрагменте кода изменяет на VARCHAR(10240) тип данных в столбце params. Ее выполнение не приводит к изменению или удалению данных, при условии, что данные в таблице соответствуют новому типу. Как правило, размеры столбцов увеличиваются, а не уменьшаются. В этом случае имеющиеся в них данные будут сохранены.

Вторая команда добавляет новый столбец в таблицу #__menu. Как видите, в нем установлено исходное нулевое значение с помощью атрибутов NOT NULL DEFAULT 0. После выполнения этой команды столбец client_id будет содержать нулевое (а не пустое) значение в каждой стоке данной таблицы. Обратите также внимание на то, что исходное нулевое значение в этом столбце указано без одинарных кавычек. В базе данных MySQL устанавливаемые значения можно указывать как в одинарных кавычка, так и без них, хотя первый способ более предпочтителен.

И третья команда вводит однозначный индекс в таблицу `#__menu`. Но эта команда может завершиться неудачно, если существующие данные неоднозначны. Так, если в таблице имеются две или более строки с одним и тем же значением в столбцах `client_id`, `parent_id` и `alias`, эта команда завершится неудачно выдачей сообщения об ошибке.

Команда DROP TABLE

Несмотря на всю простоту команды `DROP TABLE`, пользоваться ею следует очень аккуратно. Ведь она безвозвратно удаляет таблицу из базы данных, а также любые данные, хранящиеся в этой таблице. Ниже приведен синтаксис команды `DROP TABLE`.

```
DROP TABLE <имя таблицы>;
```

Применение программы phpMyAdmin

Обращаться к базе данных MySQL можно, конечно, из командной строки, но работать с ней намного удобнее с помощью специальной программы phpMyAdmin, распространяемой в составе таких широко известных программных пакетов LAMP для веб-серверов, как XAMPP, а также доступной на большинстве веб-узлов. Ее можно также загрузить в виде отдельного пакета по адресу `http://www.phpmyadmin.net`. Полная документация на программу phpMyAdmin имеется на веб-сайте ее разработчиков по указанному выше адресу. А ниже дается краткое введение в эту программу, где поясняется, как пользоваться ею при тестировании и отладке запросов SQL, создании сценариев DDL, резервном копировании или переносе базы данных на другой сервер.

Тестирование и отладка запросов SQL

Для запуска phpMyAdmin, как правило, достаточно щелкнуть на соответствующей ссылке, предоставляемой на рабочем веб-узле или начальной странице локального веб-узла. В зависимости от конкретной установки вам, возможно, придется зарегистрироваться под своим именем пользователя MySQL и паролем. Пример начального экрана phpMyAdmin приведен на рис. 11.1.

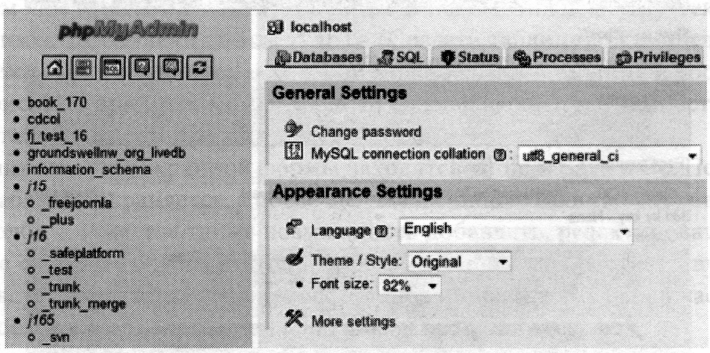


Рис. 11.1. Начальный экран программы phpMyAdmin

В левой части начального экрана показаны базы данных, доступные на данном веб-сервере. Как правило, работать с базами данных можно только по очереди. После выбора

конкретной базы данных в левой части начального экрана программы phpMyAdmin появится следующий экран, аналогичный приведенному на рис. 11.2.

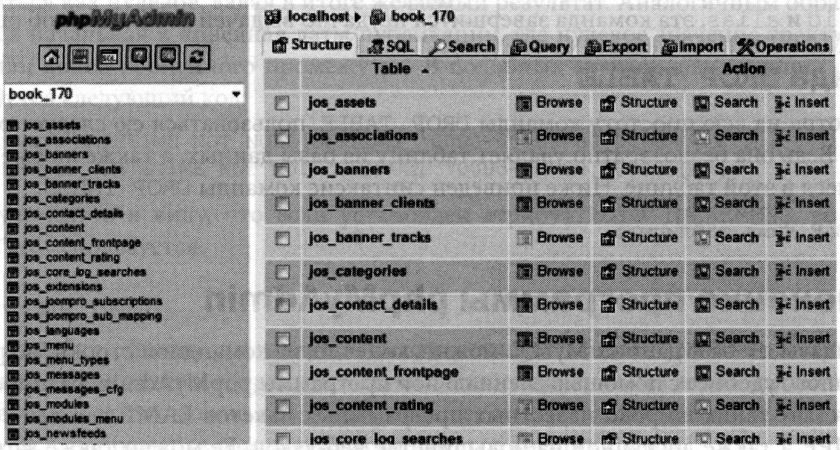


Рис. 11.2. Экран с выбранной базой данных

Как только будет выбрана конкретная база данных, в левой части экрана появятся ее таблицы. И теперь можно будет делать запросы SQL к базе данных, не указывая ее специально. Если щелкнуть на вкладке SQL справа вверху, то появится поле для ввода текста, где можно вводить и выполнять команды SQL. Так, если ввести в этом поле следующую команду:

```
SELECT * FROM jos_content
```

а затем щелкнуть на кнопке Go (Перейти) в левом нижнем углу экрана, то появится экран, аналогичный приведенному на рис. 11.3.

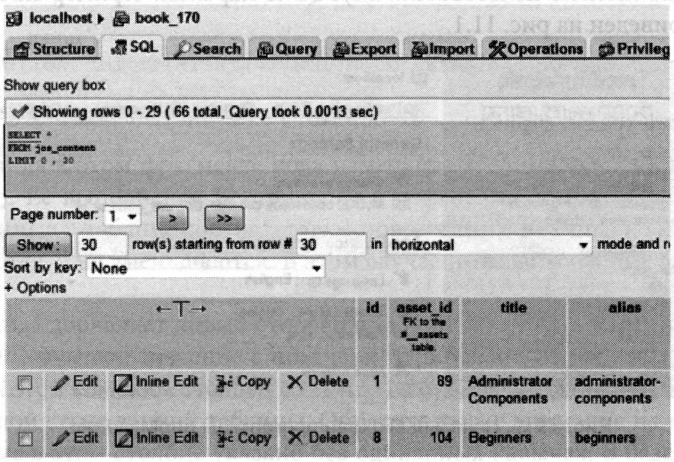


Рис. 11.3. Результаты запроса SQL

Если запрос будет обработан успешно, появится сообщение "Showing rows ..." (Показ строк ....) с общим количеством строк таблицы, возвращаемых из запроса. Кроме того, будет показана первая групп строк в таблице. А если в синтаксисе запроса обнаружится ошибка, то появится уведомляющее о ней сообщение.

В процессе работы с запросами SQL, требующимися в конкретном проекте программирования для Joomla, нередко оказывается очень полезно писать сначала запросы на SQL, а затем тестировать их описанным выше способом. Это позволяет убедиться в том, что запрос действует предполагаемым образом. Но с помощью phpMyAdmin можно также выполнять отладку запросов в Joomla. Для этого достаточно выполнить несколько действий.

- Перейдите на вкладку System экрана Global Configuration в административной части веб-сайта на Joomla и установите кнопку-переключатель Debug System (Отладка системы) в положение Yes.
- Перейдите к тому экрану, где должен выполняться нужный запрос. Запросы SQL для этого экрана будут показываться ниже обычного отображения страницы.
- Выберите и скопируйте нужный запрос из экранной области отладки и вставьте его в поле для ввода команд SQL на экране phpMyAdmin.
- Щелкните на кнопке Go, чтобы начать выполнение запроса непосредственно в phpMyAdmin.

Теперь вам будет легко и удобно работать с запросом, тестируя разные его варианты до тех пор, пока он не будет отлажен полностью. После этого вы можете вернуться в исходному коду PHP в Joomla и внести в него соответствующие изменения, чтобы правильно сформировать нужный запрос.

Создание сценариев DDL

Программа phpMyAdmin позволяет также упростить процесс написания сценариев DDL для создания или редактирования таблиц базы данных. В phpMyAdmin таблицы можно создавать или редактировать, пользуясь удобным графическим интерфейсом. Под названием последней по списку таблицы в левом нижнем углу текущего экрана phpMyAdmin имеется ссылка Create Table (Создать таблицу). Если щелкнуть на этой ссылке, появится экранная форма, в которой можно создать таблицу и составляющие ее столбцы. На рис. 11.4 приведен моментальный снимок экрана, где демонстрируется пример создания таблицы, состоящей из двух столбцов.

В правом нижнем углу экранной формы находятся кнопки Save и Go. После щелчка на кнопке Save таблица сохраняется, а после щелчка на кнопке Go в таблицу вводятся дополнительные столбцы. Имеется также возможность добавлять, редактировать или удалять столбцы в уже существующей таблице. На рис. 11.5 показаны соответствующие ссылки для изменения, удаления или добавления столбцов в таблице. А в левой части экрана под заголовком Indexes находятся кнопки для ввода и правки индексов.

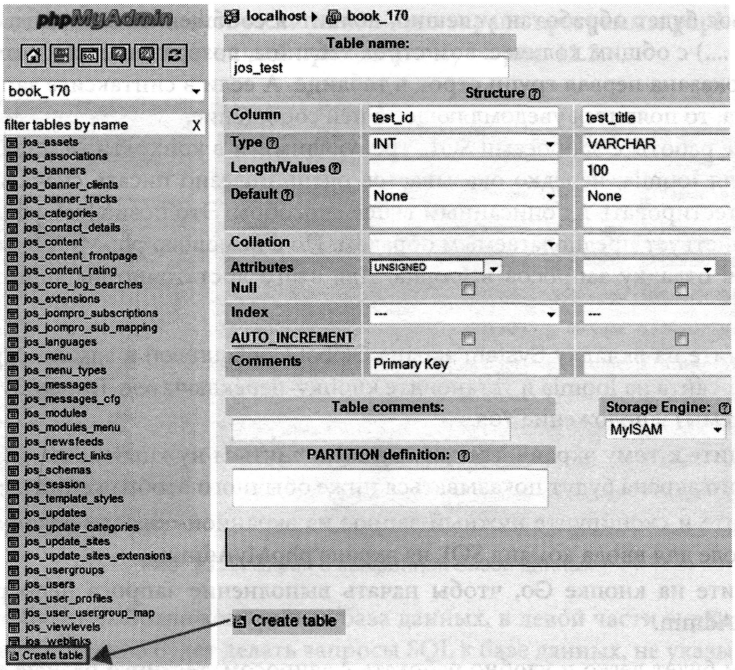


Рис. 11.4. Создание таблицы средствами phpMyAdmin

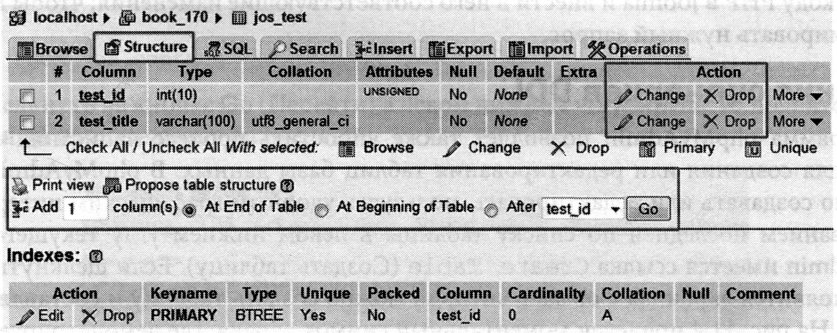


Рис. 11.5. Добавление, изменение или удаление столбцов из таблицы средствами phpMyAdmin

Как только таблица будет определена требуемым образом, в phpMyAdmin предоставляется возможность автоматически составить сценарий DDL для создания этой таблицы. Для этого достаточно щелкнуть на кнопке Export. В итоге появится экранная форма с вариантами выбора режимов быстрого (Quick) и специального (Custom) создания таблицы. Выберите сначала режим Quick, а затем вариант SQL из списка Format и щелкните на кнопке Go. В phpMyAdmin будет автоматически составлен сценарий DDL и предоставлена возможность открыть или сохранить его в текстовом файле. Созданный в итоге текстовый файл будет содержать не только сценарий DDL для создания таблицы, но и дополнительные сведения, которые вряд ли вам понадобятся. В качестве примера ниже приведен листинг такого сценария для создания таблицы `jos_test`.

```
- -
- - Структура таблицы `jos_test`
- -
CREATE TABLE IF NOT EXISTS `jos_test` (
  `test_id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT 'Primary Key',
  `test_title` varchar(100) DEFAULT NULL COMMENT 'Title',
  PRIMARY KEY (`test_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

Важное примечание

Работая с таблицами в phpMyAdmin, вы должны пользоваться конкретным префиксом, выбранным в Joomla для текущей базы данных (в приведенном выше примере — префиксом "**jos_**"). Когда же настанет момент применить тот же самый сценарий в приложении Joomla, не забудьте заменить этот префикс на "**#__**", чтобы затем он был автоматически заменен средствами Joomla на префикс, выбранный для конкретного веб-сайта.

Резервное и обычное копирование базы данных

Как упоминалось ранее в этой главе, работая с такими базами данных SQL, как, например, MySQL, совсем не обязательно знать конкретное физическое расположение файлов данных и порядок их организации. Как же в таком случае быть, если требуется сделать резервную копию базы данных или перенести ее с одного сервера на другой?

Для решения одной из этих задач можно воспользоваться средствами экспорта и импорта базы данных в phpMyAdmin. Так, если требуется сделать резервную копию базы данных, то для этой цели создается экспортный файл, содержащий ряд команд, которые могут быть выполнены в любой базе данных MySQL. Когда экспортный файл создается для всей базы данных, он содержит все команды SQL, необходимые для восстановления базы данных. К их числу относятся команды `CREATE TABLE` типа DDL, предназначенные для создания структуры таблиц базы данных, а также команды `INSERT` типа DML для заполнения этих таблиц данными.

Ниже приведена последовательность действий для экспорта всей базы данных.

- Перейдите к начальному экрану phpMyAdmin и щелкните на нужной базе данных в списке слева. В итоге появится экран, на котором перечислены все таблицы этой базы данных, как показано на рис. 11.2.
- Щелкните на вкладке Export в верхней части экрана и выберите кнопку-переключатель Quick ниже метки Export Method (Способ экспорта), а также вариант SQL из раскрывающегося списка Format. Щелкните далее на кнопке Go.
- Щелкните на кнопке Save File (Сохранить файл) и выберите имя экспортного файла, а также подходящее место для его хранения.
- Откройте созданный в итоге экспортный файл и просмотрите его. Он должен содержать команду `CREATE TABLE` для каждой таблицы базы данных, а также команду `INSERT INTO` для каждой таблицы, содержащей данные.

Воспользуйтесь далее созданным экспортным файлом, чтобы создать новую базу данных, являющуюся точной копией экспортированной базы данных. Для этого необходимо сначала создать новую базу данных, выполнив следующие действия.

- Перейдите к начальному экрану phpMyAdmin и щелкните на вкладке Databases (Базы данных) в верхней части этого экрана. В итоге появится поле Create new database (Создать новую базу данных).
- Выберите имя базы данных (например, `jos_test_copy`), а в качестве схемы сопоставления — `utf8_general_ci`, где `ci` означает "без учета регистра". Таким образом, строчные и прописные буквы будут рассматриваться как равнозначные при сортировке данных. Набранные строчные и прописные буквы будут по-прежнему отображаться правильно, но если в команде SQL указан оператор `ORDER BY`, то строчные и прописные буквы будут упорядочиваться как набранные в одном регистре. Это стандартный и наиболее предпочтительный режим работы для большинства баз данных SQL.
- Нажмите кнопку Create. В итоге должно быть выдано сообщение, уведомляющее о том, что база данных создана успешно.
- Выберите новую базу данных и нажмите кнопку Import в верхнем ряду экрана.
- Найдите экспортный файл, созданный ранее в процессе экспорта всей базы данных.
- Щелкните на кнопке Go. Некоторое время спустя появится сообщение, уведомляющее о том, что база данных успешно импортирована.
- Проверьте наличие в новой базе данных всех таблиц и данных, присутствовавших в исходной базе данных.

Формирование запросов SQL к базе данных с помощью команд DML

Для обработки данных в таблицах на языке SQL составляются запросы к базе данных. К основным командам запросов SQL относятся следующие: `SELECT`, `UPDATE`, `INSERT`, `UNION` и `DELETE`. Тема составления запросов SQL весьма обширна и сложна, поэтому ниже будут рассмотрены лишь самые основы. Но и этого должно быть достаточно, чтобы составлять большинство запросов к базе данных, требующихся в приложениях Joomla.

Запросы по команде `SELECT`

Чаще всего в приложениях Joomla применяются запросы по команде `SELECT`. Они служат для выборки данных из таблиц в базе данных. Ниже приведена простейшая структура запроса по команде `SELECT`.

```
SELECT <список столбцов и выражений>
FROM <главная таблица>
JOIN <список операторов соединения>
WHERE <условия отбора>
ORDER BY <упорядочиваемые столбцы>
LIMIT <смещение, число строк>
```

Все перечисленные выше операторы совсем не обязательно включать в запрос по команде `SELECT`. Но уж если они включаются в запрос, то должны быть непременно

перечислены в указанном выше порядке. Как поясняется далее в этой главе, преимущество класса JDatabaseQuery, применяемого для составления запросов базы данных SQL в коде приложений Joomla, заключается, в частности, в том, что в запрос можно вводить операторы в любом порядке. Но при выполнении запросов непосредственно в базе данных (например, средствами phpMyAdmin) необходимо строго придерживаться порядка следования, установленного для разных операторов SQL.

Список столбцов и выражений

В первой части запроса по команде SELECT указывается список столбцов и выражений, отображаемых в результатах запроса. Это должны быть заголовки столбцов в строках, возвращаемых из таблицы. Для обозначения таблицы, из которой извлекается столбец, служит псевдоним. Например, в запросе

```
SELECT a.id, a.name FROM jos_content AS a
```

таблица jos_content обозначается псевдонимом a. Следует иметь в виду, что все ключевые слова языка SQL принято указывать прописными буквами. И хотя в запросах SQL регистр не учитывается, тем не менее данное правило упрощает чтение запросов.

Псевдонимы можно создавать и для имен столбцов таблицы. По умолчанию имя столбца в запросе совпадает с его именем в столбце, чего, как правило, достаточно. Но в некоторых случаях требуется указывать в запросе другое имя столбца (например, при наличии в двух таблицах столбцов с одинаковыми именами или при использовании выражения для обращения к столбцу). В подобных случаях для имени столбца создается псевдоним с помощью ключевого слова AS, как показано ниже.

```
SELECT a.id AS article_id, a.name FROM jos_content AS a
```

В данном примере столбец id будет отображаться как article_id. Следует, однако, иметь в виду, что знаки точки нельзя употреблять в именах столбцов и псевдонимов (разве что заключив их в кавычки), поскольку точками псевдонимы таблиц отделяются от имен столбцов. По желанию ключевое слово AS можно опустить, и тогда оно будет подразумеваться. Например, следующий запрос равнозначен приведенному выше:

```
SELECT a.id article_id, a.name FROM jos_content a
```

Но, как правило, ключевое слово AS желательно указывать в запросе явным образом, чтобы упростить чтение самого запроса. Для указания в запросе всех столбцов таблицы служит знак звездочки (*). Например, по запросу

```
SELECT a.* FROM jos_content AS a
```

из таблицы будут возвращены все строки. В этом случае указать псевдонимы столбцов нельзя. И наконец, вместо столбцов в запрос можно включить выражения. Это нередко делается в операторе GROUP BY, о котором речь пойдет далее.

Оператор FROM

В операторе FROM перечисляются таблицы, из которых предстоит извлечь данные. Синтаксис языка SQL допускает два равнозначных способа соединения таблиц, что нередко приводит к путанице. Один из этих способов состоит в перечислении всех таблиц в операторе FROM и включении условий их соединения в операторе WHERE. А другой, более

предпочтительный способ (по крайней мере, для Joomla) состоит в применении команды JOIN. Этот способ оказывается более предпочтительным потому, что он делает более ясным назначение запроса для того, кто читает исходный код.

Когда оператор JOIN используется для соединения вторичных таблиц, в оператор FROM для запроса включается только главная таблица. А какая таблица является главной, или первичной? Как правило, это таблица, из которой требуется извлечь нечто главное: сведения о статье или пользователе. Поэтому для извлечения сведений об искомом элементе запрос обычно начинается с главной таблицы, а затем в него вводятся другие, связанные с ней таблицы, где хранятся дополнительные сведения о данном элементе. Таким образом, главная таблица является первичной для запроса. Как упоминалось ранее, с помощью ключевого слова AS обычно указывается короткий запрос, как, например, FROM jos_content AS a.

Оператор JOIN

Операторы JOIN служат для объединения столбцов из двух или более таблиц, что зачастую и делается в большинстве запросов. При построении баз данных SQL данные разных типов рекомендуется размещаться в отдельных таблицах. А это означает, что в одном запросе приходится извлекать информацию из нескольких таблиц.

Так, в Joomla данные категории обычно хранятся в отдельной таблице (например, jos_categories), тогда как в связанных с ней таблицах (например, jos_content) — лишь идентификатор категории (в виде внешнего ключа). При составлении запроса на отображение статей из таблицы jos_content нередко требуется отображать имя категории или другие сведения о ней, а не только ее идентификатор. Именно для этого таблица категорий и присоединяется к запросу. Это дает возможность отображать имя категории или любой другой столбец данной категории.

При освоении запросов SQL могут возникнуть определенные неясности в отношении операторов JOIN. Чаще всего применяются операторы внутреннего соединения (INNER JOIN) и левого соединения (LEFT JOIN), иногда еще называемого левым внешним соединением (LEFT OUTER JOIN). По умолчанию предполагается внутреннее соединение, если тип соединения не указан явно. Рассмотрим следующий простой пример запроса на отображения статей и их категорий:

```
SELECT a.title, c.title AS category_title
FROM jos_content AS a
INNER JOIN jos_categories AS c
ON a.catid = c.id
```

В данном примере отображается название каждой статьи и ее категории. Для указания псевдонимов каждой таблицы и названия категории в данном запросе используется ключевое слово AS, иначе в запросе оказались бы два столбца с одинаковым именем title. Обратите также внимание на применение в этом запросе оператора INNER JOIN. Для того чтобы добиться аналогичного результата, можно было бы просто указать оператор JOIN, поскольку операция внутреннего соединения (INNER) подразумевается в нем по умолчанию.

В последней строке рассматриваемого здесь запроса указан оператор ON, что очень важно. Ведь в нем, по существу, устанавливается взаимосвязь обеих таблиц для выполнения

данного запроса. Оператор ON *должен всегда* сопутствовать оператору JOIN, иначе предполагаемый результат не будет получен. Так, если опустить в данном примере оператор ON, то в конечном итоге будут получены строки, состоящие из комбинации строк, извлекаемых из двух соединяемых таблиц. Например, в выборочных данных для версии Joomla 2.5 насчитывается 66 строк в таблице jos_content и 69 строк в таблице jos_categories, и если опустить в данном запросе оператор ON, то в результате его выполнения получится 4554 строки!

В операторе ON нередко указывается внешний ключ для установления связи между двумя таблицами. В данном примере указан внешний ключ a.catid к таблице jos_categories. Имеется немало других примеров установления связи между двумя таблицами по внешнему ключу в базе данных Joomla, включая связи между таблицами jos_banners и jos_categories, jos_banners и jos_banner_clients, jos_contact_details и jos_users и т.д.

Рассмотрим далее отличие внутреннего соединения (INNER JOIN) от левого (LEFT JOIN). При внутреннем соединении требуется, чтобы запрашиваемые строки существовали в обеих таблицах. В предыдущем примере у каждой статьи, перечисляемой в запросе, имеется своя категория, что всегда справедливо для обычной базы данных в Joomla, где каждая статья относится только к одной категории. Поэтому в данном примере внутреннее и левое соединения дают один и тот же результат.

Обратимся теперь к другому примеру. Допустим, требуется получить список статей, независимо от того, помечены ли они для блога по избранным статьям на первой странице веб-сайта. В таблице jos_content_frontpage содержится строка для каждой избранной статьи, включая ее идентификатор и порядковый номер. Ниже приведен первый вариант запроса к базе данных.

```
SELECT a.title, f.ordering
FROM jos_content AS a
INNER JOIN jos_content_frontpage AS f
ON a.id = f.content_id
```

Это запрос на отображение названия статьи и порядка ее расположения на первой странице. Указав в запросе только внутреннее соединение, мы включаем в запрос только статьи со строкой из таблицы jos_content_frontpage. Используя выборочные данные для версии Joomla 2.5, мы в итоге получим следующие строки:

title	ordering
Beginners	2
Professionals	4
Joomla!	1
Upgraders	3

А теперь посмотрим, что произойдет, если воспользоваться в том же самом запросе левым соединением, как показано ниже.

```
SELECT a.title, f.ordering
FROM jos_content AS a
LEFT JOIN jos_content_frontpage AS f
ON a.id = f.content_id
```

В результате выполнения этого запроса будут получены все статьи из таблицы jos_content, даже если в таблице jos_content_frontpage для них отсутствует строка. В этом случае в столбце f.ordering окажется пустое значение (NULL), означающее отсутствие статьи. Данный запрос можно изменить таким образом, чтобы указать, помечена ли статья для отображения на первой странице, воспользовавшись оператором CASE языка SQL, как показано ниже.

```
SELECT a.title,
CASE WHEN (f.content_id IS NULL) THEN 'No' ELSE 'Yes' END AS front_page
FROM jos_content AS a
LEFT JOIN jos_content_frontpage AS f
ON a.id = f.content_id
```

Во второй строке этого запроса используется выражение, введенное в команду SELECT для составления списка статей. Оператор CASE позволяет отображать различные значения в зависимости от результатов проверок разных логических условий. В данном случае проверяется, является ли пустым значение в столбце f.content_id. Если оно окажется пустым, значит, статья отсутствует в таблице jos_content_frontpage, а следовательно, в результатах данного запроса напротив статьи отображается значение "No". В противном случае статья присутствует в таблице jos_content_frontpage, и поэтому в результатах данного запроса напротив статьи отображается значение "Yes". Ниже приведены пять первых строк, возвращаемых из базы данных в результате выполнения данного запроса.

Title	front_page
Administrator Components	No
Beginners	Yes
ArchiveModule	No
Banner Module	No
Article CategoriesModule	No

Для извлечения данных из каждой требующейся таблицы в запросе можно указать сколько угодно операторов JOIN. При этом псевдоним каждой таблицы должен быть однозначным.

Оператор WHERE

С помощью этого оператора можно отфильтровать строки таблицы, чтобы включить в запрос только те из них, которые удовлетворяют заданным условиям. Операторы WHERE могут быть как простыми, так и сложными. В приведенном ниже примере оператор WHERE введен в запрос, чтобы включить в него только опубликованные статьи.

```
SELECT a.title, c.title AS category_title
FROM jos_content AS a
INNER JOIN jos_categories AS c
ON a.catid = c.id
WHERE a.state = 1
```

Несколько выражений в операторе WHERE можно связать логическим оператором AND или OR. Так, в следующем примере по запросу отображаются лишь те статьи, которые опубликованы *и* имеют ненулевую дату публикации в столбце publish_down:

```
SELECT a.title, c.title AS category_title
FROM jos_content AS a
INNER JOIN jos_categories AS c
ON a.catid = c.id
WHERE a.state = 1
AND a.publish_down != '0000-00-00 00:00:00'
```

Напомним, что по умолчанию в столбце publish_down устанавливается значение '0000-00-00 00:00:00'. Это означает, что статья вообще не снята с публикации. В выборочных данных ни одна из статей не удовлетворяет этому условию, поэтому для проверки данного запроса следует перейти к компоненту Article Manager и ввести дату окончания публикации статьи. При повторном выполнении запроса эта статья должна быть отображена.

Если выражения связываются в операторе WHERE логическим оператором OR, то по запросу из таблицы извлекаются *любые* строки, удовлетворяющие заданным условиям. Так, в приведенном ниже примере по запросу получаются строки таблицы с псевдонимом создания "Parks Webmaster" *или* названием категории "Plugins".

```
SELECT a.title, c.title AS category_title, a.created_by_alias
FROM jos_content AS a
INNER JOIN jos_categories AS c
ON a.catid = c.id
WHERE a.created_by_alias = 'Parks Webmaster'
OR c.title = 'Plugins'
```

Следует иметь в виду, что в операторе WHERE могут быть указаны столбцы из любой таблицы. Если же выражения в операторе WHERE требуется связать обоими логическими операторами, AND и OR, то они заключаются в круглые скобки. Так, в следующем примере в запрос включены строки таблицы из предыдущего примера, но при условии, что название статьи должно начинаться с букв **"Ed"** *или* **"Au"**:

```
SELECT a.title, c.title AS category_title, a.created_by_alias
FROM jos_content AS a
INNER JOIN jos_categories AS c
ON a.catid = c.id
WHERE
  (a.created_by_alias = 'Parks webmaster'
  OR c.title = 'Plugins')
AND
  (a.title LIKE 'Ed%'
  OR
  a.title LIKE 'Au%')
```

Следует заметить, что выражение LIKE допускает сопоставление с метасимвольным шаблоном, где знак процента обозначает совпадение с любым символом. Если бы в данный запрос было введено выражение a.title LIKE '%abc%', то сопоставление было бы сделано с любым столбцом, содержащим буквы 'abc' где угодно в тексте, тогда как выражение a.title LIKE '%abc' означало бы сопоставление с любым текстом, оканчивающимся буквами 'abc'.

Оператор ORDER BY

Оператор ORDER BY служит в запросе для сортировки строк таблицы в нужном порядке. С помощью выражений ASC и DESC можно указать обычный (по восходящей от **A** до **Z**) или обратный (по нисходящей от **Z** до **A**) порядок сортировки соответственно. Для сортировки можно указать несколько столбцов, разделив их запятыми. Например, в следующем операторе ORDER BY:

```
ORDER BY a.created_by_alias ASC, d.created DESC
```

строки, извлекаемые из таблицы по запросу, сортируются в алфавитном порядке по столбцу created_by_alias. Если же у двух или более статей окажется одинаковое значение в столбце created_by_alias, они будут упорядочены по дате и времени создания, причем первой будет отображена самая свежая статья.

Оператор LIMIT

В базе данных MySQL имеется возможность ограничить строки, возвращаемые по запросу из таблицы, с помощью оператора LIMIT. Синтаксис LIMIT xx, yy этого оператора позволяет ограничить запрос первыми строками yy, начиная со строки xx + 1.

Так, если требуется ограничить запрос возвращаемыми из таблицы строками **6–15**, в него следует ввести оператор LIMIT 5, 10. В этом случае пять первых строк пропускаются, а по запросу из таблицы возвращаются следующие десять строк. А если ввести в запрос оператор LIMIT 0, 10, то из таблицы будут возвращены первые десять строк. Если же строки пропускать не нужно, то достаточно опустить первый аргумент в данном операторе, написав его следующим образом: LIMIT 10.

Формирование составных запросов с помощью оператора GROUP BY

В обычных запросах по команде SELECT строки возвращаются из таблиц на том уровне детализации, на каком запрашиваются таблицы. Если же требуется подытожить данные, то для этой цели служит оператор GROUP BY. Запрос, включающий в себя оператор GROUP BY, называется *составным*. Если оператор GROUP BY присутствует в запросе, он должен следовать после оператора WHERE и перед оператором ORDER BY.

Применение оператора GROUP BY оказывает влияние на перечисление столбцов в списке. В частности, перечислить в списке можно, во-первых, столбцы, используемые для группирования, а во-вторых, столбцы, указываемые в составных выражениях. И в этом есть свой смысл. Ведь в составном запросе несколько строк таблицы сводятся в одну, и поэтому в базе данных должно быть известно, каким образом подытоживать значения. Именно это и указывается в составных выражениях.

Обратимся к конкретному примеру. Допустим, требуется получить число статей по каждому значению в столбце created_by_alias. Для этой цели служит приведенный ниже запрос.

```
SELECT a.created_by_alias, COUNT(*) AS article_count
FROM jos_content AS a
GROUP BY a.created_by_alias
```

По этому запросу возвращается следующий результат:

```
created_by_alias          article_count
                                     12
Fruit Shop Webmaster                  4
Joomla!                              40
Joomla! 1.5                           1
Parks webmaster                       9
```

В данном запросе указаны два столбца: `created_by_alias` и обобщающая функция `COUNT`. В качестве аргумента этой функции указывается знак звездочки (`"*"`). Эта функция подсчитывает количество строк в таблице для каждого значения из столбца `created_by_alias` в отдельности. Обратите внимание на то, что в 12 статьях искомое значение в столбце `created_by_alias` не обнаружено, и поэтому они отображаются как пустые.

В MySQL и других базах данных SQL предоставляется целый ряд полезных обобщающих функций, в том числе `MIN()`, `MAX()`, `AVG()`, `SUM()`, `COUNT()` и `COUNT(DISTINCT)`. Допустим, требуется видоизменить запрос таким образом, чтобы показать самые последние по дате написания статьи каждого автора. Для этой цели можно было бы ввести в данный запрос следующий столбец:

```
MAX(created) as last_create_date
```

А если, например, требуется исключить из запроса статьи, где отсутствует значение в столбце `created_by_alias`, то в запрос можно ввести оператор `WHERE`, чтобы проверить наличие какого-нибудь значения в этом столбце. И здесь приходится вдаваться в тонкости обработки данных в столбцах таблиц по запросам SQL. Если обратиться к определению данного столбца в базе данных, то можно обнаружить, что пустые значения `NULL` в нем не допускаются и что по умолчанию в нем сохраняется пустая символьная строка, обозначаемая двумя одиночными кавычками без пробела между ними (`''`). В силу этого проверять наличие пустых значений `NULL` в данном столбце не имеет никакого смысла — они просто в нем отсутствуют. Но в то же время можно проверить наличие в нем значения, которое больше пустой символьной строки, как показано в приведенном ниже запросе.

```
SELECT a.created_by_alias, COUNT(*) as article_count,
MAX(a.created) as last_create_date
FROM jos_content AS a
JOIN jos_categories AS c
ON a.catid = c.id
WHERE a.created_by_alias > ''
GROUP BY a.created_by_alias
```

В этом запросе пропускаются 12 статей, где искомое значение в столбце `created_by_alias` не обнаружено. Но что, если требуется показать только те строки таблицы, где имеются хотя бы две статьи? Такое условие не удастся задать в операторе `WHERE`, поскольку каждая строка проверяется в таблице по очереди. Ведь неизвестно, сколько статей может быть в конечном итоге получено по данному запросу для каждого значения в столбце `created_by_alias`. Для выхода из подобного затруднительного положения в языке SQL предоставляется оператор `HAVING`, который может быть введен в запрос непосредственно после оператора `GROUP BY` и до оператора `ORDER BY`. Этот оператор похож на оператор

WHERE, за исключением того, что в нем проверяются значения обобщающих функций. И для того чтобы ограничить предыдущий запрос только теми строками таблицы, в которых имеются две и более статьи, его можно видоизменить следующим образом:

```
SELECT a.created_by_alias, COUNT(*) as article_count,
MAX(a.created) as last_create_date
FROM jos_content AS a
JOIN jos_categories AS c
ON a.catid = c.id
WHERE a.created_by_alias > ''
GROUP BY a.created_by_alias
HAVING COUNT(*) >= 2
```

Теперь по данному запросу будут показаны только те строки таблицы, где число статей равно двум и больше.

В операторе GROUP BY можно также указывать обобщающие функции. Допустим, требуется упорядочить строки, извлекаемые из таблицы, указав сначала в списке строки с наибольшим числом статей. Для этого достаточно ввести в конце запроса следующий оператор:

```
ORDER BY COUNT(*) DESC
```

В итоге из таблицы будут извлечены строки, упорядоченные в списке от наибольшего до наименьшего числа статей.

Запросы по команде UPDATE

Запросы, формируемые по команде UPDATE, позволяют видоизменить содержимое столбцов таблицы. Ниже приведена общая форма запроса, составляемого по команде UPDATE.

```
UPDATE <таблица>
SET <список имен столбцов и их значений>
WHERE <условия отбора>
```

Например, для поиска и замены значений в столбце created_by_alias можно составить следующий запрос:

```
UPDATE jos_content
SET created_by_alias = 'Site Administrator'
WHERE created_by_alias = 'Joomla!'
```

Для того чтобы обновить содержимое нескольких столбцов, их имена и присваиваемые им значения указываются в запросе через запятую. Например, в приведенном ниже запросе, помимо столбца created_by_alias, обновляется также столбец date_created, которому присваивается дата создания статьи 1 января 2014 года.

```
UPDATE jos_content
SET created_by_alias = 'Site Administrator',
created = '2014-01-01 12:00:00'
WHERE created_by_alias = 'Joomla!'
```

Синтаксис оператора WHERE в запросах по команде UPDATE практически ничем не отличается от соответствующего синтаксиса в запросах по команде SELECT.

Запросы по команде INSERT

Запросы, формируемые по команде INSERT, служат для ввода новых строк в таблицы базы данных. Ниже приведена общая форма запроса, составляемого по команде INSERT.

```
INSERT INTO <таблица>
(<столбец 1, столбец 2, . . . >)
VALUES (<список значений для строки 1>), (<список значений для строки 2>)
```

Если список столбцов в данном запросе опущен, то предполагается, что включенный в него список значений вводится во все столбцы в том порядке, в каком они определены в таблице. В файлах сценариев базы данных joomla.sql и sample_data.sql из папки installation/sql/mysql можно обнаружить немало примеров применения запросов по команде INSERT. А в экспортном файле, созданном ранее в этой главе, имеется следующий запрос по команде INSERT:

```
INSERT INTO `jos_content_frontpage` (`content_id`, `ordering`) VALUES
(8, 2),
(35, 4),
(24, 1),
(50, 3);
```

Обратите внимание на то, что в данном запросе имена таблицы и ее столбцов заключены в обратные кавычки. Как пояснялось ранее, данное правило рекомендуется соблюдать, но это и не обязательно, кроме тех случаев, когда в имени используется ключевое слово или специальный символ вроде точки или пробела.

Следует, однако, избегать дублирования ключей или индексов при вводе строк в таблицу. Так, если определить первичный ключ, а затем попытаться ввести в таблицу строку с первичным ключом, дублирующим ключ в уже имеющейся строке, то в базе данных возникнет ошибка. А если имеется первичный ключ, определенный как AUTO_INCREMENT, то для этого ключа можно указать пустое значение NULL, и база данных автоматически поместит следующее доступное значение в столбце таблицы. Именно это обычно и требуется при вводе строки в таблицу уже имеющейся базы данных, где заранее неизвестно, какие именно первичные ключи уже находятся в таблице. Как пояснялось ранее, при создании новой таблицы, которая исходно пуста, можно указать столбец id. Именно так и делается в сценариях из файлов joomla.sql и sample_data.sql, а также в сценарии резервного копирования базы данных.

Запросы по команде DELETE

Запросы, формируемые по команде DELETE, служат для безвозвратного удаления строк из таблиц базы данных. Ниже приведена общая форма запроса, составляемого по команде DELETE.

```
DELETE FROM <имя таблицы>
WHERE <условия отбора>
```

В качестве примера ниже приведен запрос на удаление строк из таблицы jos_content_frontpage.

```
DELETE FROM jos_content_frontpage
WHERE content_id = 35
```

Синтаксис оператора WHERE в данном запросе остается таким же, как и в запросах по команде SELECT. Следует, однако, иметь в виду, что если в запросе по команде DELETE опустить оператор WHERE, то из таблицы будут удалены *все* строки.

Запросы по команде UNION

Запросы, формируемые по команде UNION, позволяют объединить сходные столбцы из двух и более таблиц в одном запросе. Такие запросы редко используются в приложения Joomla. Ниже приведена общая форма запроса, составляемого по команде UNION.

```
<SELECT запрос 1>
UNION
<SELECT запрос 2> . . .
```

Например, для просмотра списка всех элементов веб-ссылок и лент новостей можно составить следующий единый запрос по команде UNION:

```
SELECT 'Weblink' as type, w.title AS title
FROM jos_weblinks w
UNION
SELECT 'Newsfeed' as type, n.name as title
FROM jos_newsfeeds n
```

В запросе по команде UNION имена столбцов для каждого запроса, выполняемого по команде SELECT, должны быть указаны одинаково и в одном и том же порядке.

Выражения в запросах

Вместо имен столбцов в большинстве запросов могут быть введены выражения. В выражении обычно указывается имя столбца, чтобы обозначить место для записи значения. Характерным тому примером служит оператор SET в запросе по команде UPDATE или INSERT.

Как упоминалось выше, в MySQL и других базах данных SQL предоставляется целый ряд полезных вспомогательных функций. К их числу относятся функции обработки символьных строк: UPPER(), LOWER(), LENGTH(); функции обработки даты и времени: DATE_ADD(), DATE() и DATEDIFF(); а также функция CASE(), обеспечивающая логику выполнения запроса по условию. С полным перечнем выражений, доступных в MySQL, можно ознакомиться в документации на эту базу данных по адресу http://dev.mysql.com/doc/refman/5.1/en/functions.html.

Разработка структуры таблиц

Составление структуры таблиц базы данных является важной стадией разработки расширения Joomla. Общий принцип составления структуры базы данных заключается в следующем: для каждого типа объекта-сущности в базе данных должна быть выделена отдельная таблица, причем в данной таблице должен находиться объект-сущность только одного типа. Рассмотрим в качестве примера базу данных для учебного заведения. Она предназначена для отслеживания учащихся, преподавателей и их классов. Таблицы этой базы данных можно разделить на группы по типам информации, которая будет храниться в них. Следовательно, для такой базы данных можно выделить следующие типы таблиц.

- **Справочная таблица.** Содержит статическую информацию об объекте-сущности, например, о заказчиках, контактах и статьях. Зачастую справочные таблицы сопровождаются самими пользователями, вводящими и редактирующими их строки, например, в специальной экранной форме.
- **Таблица соответствий.** Служит для приведения одних таблиц в соответствие с другими, например, пользователей с группами или учащихся с классами. Как правило, таблицы соответствий сопровождаются системой в зависимости от конкретных действий пользователя.
- **Таблица предыстории.** Служит для записи подробных сведений, например, журналов регистрации, предыстории закупок или же истории классов. Обычно таблицы предыстории автоматически сопровождаются системой.

Ниже будут рассмотрены характерные примеры всех перечисленных выше типов таблицы базы данных.

Справочные таблицы

Начнем рассмотрение примера базы данных для учебного заведения со следующих четырех справочных таблиц.

- **Учащиеся.** В этой таблице содержатся все сведения по каждому учащемуся, в том числе его Ф.И.О., домашний адрес, возраст и год обучения. Но в эту таблицу не входят такие динамически изменяющиеся данные, как, например, пройденные классы и этапы обучения.
- **Преподаватели.** В этой таблице содержатся все статические данные по каждому преподавателю, в том числе его Ф.И.О., домашний адрес, отделение и ставка заработной платы. И в эту таблицу не входят динамически изменяющиеся или исторические данные.
- **Классы.** В этой таблице содержатся все сведения о каждом классе, в том числе название и описание класса, отделение, время сбора, номер учебного помещения, даты проведения мероприятий и максимально допустимое количество учащихся.
- **Отделения.** В этой таблице содержатся все сведения о каждом отделении, в том числе название и описание отделения.

Структура каждой из перечисленных выше таблиц довольно проста. В одной строке таблицы представлены сведения об одном учащемся, преподавателе, классе или отделении. Пользователи системы, как правило, вводят или добавляют строки в эти таблицы непосредственно с помощью специальных форм.

Поля ключей и внешние ключи

Каждый объект-сущность целесообразно обозначить в перечисленных выше таблицах с помощью идентификатора в столбце `id`. В Joomla это обычно делается автоматическим приращением целочисленного значения первичного ключа, как пояснялось ранее в этой главе. В данном примере каждый преподаватель относится лишь к одному конкретному отделению, и поэтому в таблицу преподавателей можно ввести столбец как поле ключа для таблицы отделений (например, с идентификатором отделения). Если же из таблицы отделений потребуется извлечь другую информацию, это можно будет сделать, добавив в

запрос оператор JOIN для соединения таблиц по данному столбцу. А если в таблицах предполагается большое количество строк (например, более 1000), в таком случае их, вероятнее всего, придется проиндексировать по столбцам с внешними ключами, чтобы ускорить обработку запросов, в которых соединение таблиц осуществляется по этим столбцам.

Таблицы соответствий

Внешние ключи отлично подходят для связывания одной строки из одной таблицы с одной строкой в другой таблице. Но иногда требуется установить соответствие нескольких строк из одной таблицы с несколькими строками в другой таблице. Рассмотрим в данном примере взаимосвязь между таблицами учащихся и классов. Каждый учащийся может пройти обучение во многих классах, а каждый класс может быть назначен многим учащимся. Это так называемое отношение "многие ко многим". Такое отношение обычно реализуется с помощью таблицы соответствий. В данном случае можно создать таблицу соответствий student_class_map с двумя столбцами student_id и class_id, а также первичный ключ, объединяющий оба этих столбца. Иными словами, отдельный класс может быть назначен конкретному учащемуся только один раз.

Всякий раз, когда учащийся вводится в класс, система автоматически добавляет в таблицу соответствий строку со столбцами идентификаторов учащегося и класса. Если же учащийся пропускает класс, эта строка может быть удалена автоматически. С помощью таблицы соответствий можно легко выполнить два вида запросов. В частности, список учащихся в одном классе (class id = 123) можно получить по следующему запросу:

```
SELECT c.class_name, s.student_name
FROM student_class_map AS m
INNER JOIN classes AS c ON m.class_id = c.id
INNER JOIN students AS s ON m.student_id = s.id
WHERE m.class_id = 123
ORDER BY s.student_name;
```

Аналогично список всех классов, на обучение в которых записан учащийся (id = 234), может быть получен по следующему запросу:

```
SELECT s.student_name, c.class_name
FROM student_class_map AS m
INNER JOIN classes AS c ON m.class_id = c.id
INNER JOIN students AS s ON m.student_id = s.id
WHERE m.student_id = 234
ORDER BY c.class_name;
```

Следует заметить, что таблица соответствий student_class_map служит в качестве главной в обоих приведенных выше запросах, хотя отображаемые столбцы извлекаются из других таблиц. Дело в том, что для каждой строки из этой таблицы по запросу требуется получить лишь одну строку. Аналогичным образом можно создать таблицу соответствий для преподавателей и классов. Она также будет отражать отношение "многие ко многим". Аналогично справочным таблицам, таблицы соответствий можно проиндексировать по столбцам class_id и student_id, чтобы ускорить обработку запросов, в которых эти столбцы указываются как внешние ключи.

Таблицы предыстории

К третьему типу относятся таблицы предыстории, называемые иначе таблицами транзакций. В данном примере одна такая таблица должна содержать предысторию классов, в которых учащиеся прошли полное обучение. Как правило, для данного типа таблицы вполне подходит первичный ключ с атрибутом AUTO_INCREMENT. Частичная структура таблицы предыстории классов, которые окончили учащиеся, может выглядеть так, как показано ниже.

- Столбец history_id: атрибуты INT, AUTO_INCREMENT, PRIMARY KEY.
- Столбец student_id: внешний ключ к таблице учащихся.
- Столбец class_id: внешний ключ к таблице классов.
- Столбец semester: значение типа varchar(6) (формат: yyyyxx, где yyyy — учебный год, xx — 01, 02, 03, т.е. номер семестра, четверти или иного периода учебного года в зависимости от вида учебного заведения).
- Столбец date_completed: дата окончания.
- Столбец class_grade: значение типа varchar(2).
- Столбец comments: значение типа varchar(1024).

По окончании каждого класса, когда выставляется итоговая оценка, в таблицу предыстории классов вводится строка по каждому учащемуся и классу. Таким образом, табель или справку об успеваемости учащегося (т.е. перечень оконченных им классов) можно получить по следующему запросу:

```
SELECT c.name, h.semester, h.class_grade
FROM student_class_history AS h
INNER JOIN students AS s ON s.id = h.student_id
INNER JOIN classes AS c ON c.id = h.class_id
WHERE h.student_id = 234
ORDER BY h.semester ASC, c.name ASC
```

Аналогичную таблицу предыстории можно было бы создать и для записи классов, которые вел каждый преподаватель. Эту таблицу было бы целесообразно проиндексировать по столбцам student_id и class_id. Соответствующие запросы можно было бы составить и для отбора строк из этой таблицы по столбцу semester_id, а следовательно, и данный столбец стоило бы проиндексировать. Это же относится и к столбцу date_completed, если по нему предполагается делать запросы.

Работа с базой данных в среде Joomla

Итак, рассмотрев основы формирования запросов SQL к базе данных, перейдем к вопросам составления и обработки этих запросов в среде Joomla. В приложениях Joomla чаще всего приходится иметь дело с запросами по команде SELECT. Что же касается запросов SQL по командам INSERT, UPDATE и DELETE, то они нередко формируются в классе JTable автоматически. В рассмотренном ранее примере компонента подписок такие запросы к таблице #__joompro_subscriptions формировались автоматически, а для обращения к таблице соответствий #__joompro_sub_mapping запросы по командам INSERT, UPDATE и DELETE приходилось составлять вручную. Для составления запросов SQL в Joomla служит класс JDatabaseQuery, а для их выполнения и возврата результатов — класс JDatabase.

Применение класса JDatabaseQuery

Класс JDatabaseQuery был внедрен в версии Joomla 1.6 с целью упростить составление запросов SQL, а в версии Joomla 2.5 он был дополнен классами, потребовавшимися для поддержки нескольких баз данных. К их числу относятся классы, предназначенные для каждого типа поддерживаемых баз данных, в том числе класс JDatabaseQueryElementMySQLi для драйвера MySQLi, а также класс JDatabaseQuerySQLSrv для сервера базы данных Microsoft SQL Server. В этих классах учитываются мелкие отличия в синтаксисе языка SQL для разных типов поддерживаемых баз данных. Но для разработчиков самым большим облегчением является то обстоятельство, что они могут составлять запросы, которые становятся *автоматически* совместимыми со всеми поддерживаемыми базами данных, при условии, что они формируются средствами класса JDatabaseQuery с соблюдением нескольких простых правил! Запросы можно по-прежнему составлять и вручную, но в этом случае они не будут совместимы с несколькими базами данных.

Создание объекта типа JDatabaseQuery

Объект запроса создается с помощью метода getQuery() из класса JDatabase. Например, в следующей строке кода создается пустой объект запроса:

```
$query = JFactory::getDbo()->getQuery(true);
```

Благодаря указанию логического значения true в качестве аргумента при вызове метода getQuery() любой предыдущий запрос стирается и для составления сызнова предоставляется пустой запрос, что обычно и требуется. Напомним, что объект типа JDatabase создается при вызове метода JFactory::getDbo(), возвращающего тип объекта базы данных для настроенного драйвера базы данных. В свою очередь, у объекта базы данных появляются средства для создания подходящего объекта типа JDatabaseQuery для настроенной базы данных.

Следует, однако, иметь в виду, что для создания объекта пустого запроса следующая строка кода не подходит:

```
$query = new JDatabaseQuery(); // этот код непригоден!
```

Подобным способом нельзя создать объект подходящего типа для драйверов разных баз данных, и поэтому приведенный выше код будет действовать некорректно в работе с разными базами данных.

Формирование запросов на выборку средствами класса JDatabaseQuery

Если знать, каким образом на языке SQL составляются запросы на выборку по команде SELECT, то сделать это средствами класса JDatabaseQuery не составит большого труда. В табл. 11.1 перечислены методы из класса JDatabaseQuery для каждого оператора, используемого в запросе по команде SELECT.

Как пояснялось ранее, порядок следования различных операторов в запросах SQL жестко установлен. Но в классе JDatabaseQuery объект запроса может быть построен в любом порядке. Когда же запрос передается базе данных, его операторы располагаются в нужном порядке с помощью метода __toString() из класса JDatabaseQuery. Так же

просто в запрос можно ввести и дополнительные столбцы. Например, запрос может быть начат со следующего списка столбцов:

```
$query->select('a.id, a.title');
```

Таблица 11.1. Методы из класса JDatabaseQuery для составления запросов по команде SELECT

Оператор запроса по команде SELECT	Метод из класса JDatabaseQuery	Пример указания аргумента
Список столбцов	select()	'a.title, c.title AS category_title'
FROM	from()	'#__content AS a'
INNER JOIN	innerJoin()	'#__categories AS c ON a.catid = c.id'
LEFT JOIN	leftJoin()	'#__categories AS c ON a.catid = c.id'
WHERE	where()	'a.published = ' . (int) $published
ORDER BY	order()	$db->getEscaped($orderCol.''.$orderDirn)
GROUP BY	group()	'c.id'
LIMIT	Реализуется в классе JDatabase	$dbo->setQuery($query, $limitstart, $limit);

А в дальнейшем, вероятнее всего, после ввода в запрос таблицы категорий его можно дополнить столбцами, как показано ниже.

```
$query->select('c.id AS catid, c.title AS cat_title');
```

Это дает возможность составить запрос таким образом, чтобы его легко было понять и чтобы он органично вписывался в последовательность выполнения кода программы. То же самое справедливо и для методов соединения. В частности, для внутреннего и левого соединения в рассматриваемом здесь классе имеются отдельные методы, которым в качестве аргумента передается условие, определяемое оператором ON. Соединения таблиц осуществляются по очереди, а по желанию в данном месте кода можно даже ввести столбцы из присоединенной таблицы.

В методе where() можно указать в качестве второго аргумента связывающий логический оператор AND или OR, причем первый из них предполагается по умолчанию. Если же требуется составить сложные выражения WHERE, например, с условием OR, вложенным в условие AND, это делается в круглых скобках, как и в самом операторе WHERE языка SQL. Например, для воспроизведения следующего оператора WHERE языка SQL:

```
WHERE
  (a.created_by_alias = 'Parks Webmaster'
  OR c.title = 'Plugins')
AND
  (a.title LIKE 'Ed%'
  OR
  a.title LIKE 'Au%')
```

можно написать такой код:

```
$query->where('(a.created_by_alias = 'Parks Webmaster'
    OR c.title = 'Plugins')');
$query->where('(a.title LIKE 'Ed%' OR a.title LIKE 'Au%')');
```

В этом примере кода вложенные условия OR передаются методу where() в качестве первого аргумента, тогда как логический оператор AND предполагается по умолчанию. Обратите внимание на то, что для соблюдения правильного порядка вычисления выражений служат круглые скобки. В методах select() и order() имя каждого столбца или выражение отделяется запятой. Так, в методе order() порядок сортировки указывается с помощью выражений ASC или DESC, как и в коде запроса SQL.

Защита от злоумышленников

Пользуясь классом JDatabaseQuery для составления запросов, не следует забывать и о защите от внесения злоумышленниками злонамеренного кода. Так, если значение, применяемое в операторе WHERE или ORDER BY, происходит из ненадежного источника, например, HTML-формы или URL (злоумышленники могут воспользоваться и тем и другим для внесения злонамеренного кода), это значение следует подвергнуть дополнительной обработке, прежде чем использовать его в запросе. И если значение в операторе WHERE должно быть целым, его следует непременно привести к целому с помощью оператора (int). В приведенном ниже примере заранее известно, что значение переменной $catid должно быть целым, и для этого принимаются специальные меры.

```
$query->where('a.catid = ' . (int) $catid);
```

Если же значение является текстом, для заключения его в кавычки и "экранирования" любых специальных символов следует воспользоваться методом quote() из класса JDatabase. Экранирование текста означает проверку в нем наличия символов, имеющих специальное значение в конкретном языке программирования (в данном случае SQL). Когда значение экранируется, любые специальные символы отображаются таким образом, что они теряют свое специальное значение в данном языке программирования. Например, в SQL для этой цели служит обратная косая черта (\), помещаемая перед символами одиночных или двойных кавычек. Это дает возможность отобразить сами кавычки, но в то же время уведомить базу данных не интерпретировать их как специальный символ. Экранирование текстовых значений не дает злоумышленнику возможность воспользоваться кавычками или другими специальными символами с целью исказить смысл запроса и внести в него свой собственный запрос SQL. Как было показано в предыдущих главах, экранировать текст можно и по отношению к языку HTML или JavaScript.

Допустим, в операторе WHERE используется переменная $created_by, которая происходит из ненадежного источника. В таком случае в коде можно принять следующие меры защиты:

```
$query->where('a.created_by_alias = ' . $db->quote($created_by));
```

Если злоумышленник попытается ввести злонамеренный код в переменной $created_by, он будет обезврежен в методе quote(), обезопасив тем самым запрос. А если переменная используется в операторе WHERE вместе с метасимволом, необходимо принять меры к тому, чтобы правильно расположить кавычки. Допустим, в запрос требуется ввести следующий оператор WHERE:

```
WHERE a.title LIKE '%abc%'
```

где значение 'abc' берется из введенной пользователем переменной $search. В таком случае знаки процентов следует передать в качестве аргумента методу quote(), как показано ниже.

```
$query->where('a.title LIKE ' . $db->quote('%' . $search . '%'));
```

Таким образом, кавычки отделяются от знаков процентов, а значение переменной $search все равно экранируется.

В тех случаях, когда ненадежные данные передаются оператору ORDER BY, необходимо убедиться в том, что передаваемые значения являются действительными именами столбцов. В противном случае в коде запроса SQL возникнет ошибка, способная раскрыть информацию о веб-сайте потенциальным злоумышленникам. Один из способов борьбы с подобными нарушениями защиты обсуждался в главе 10 и состоит в том, чтобы создать сначала массив действительных имен столбцов, а затем проверить введенные данные по этому списку имен.

Связывание методов в цепочку

Методы из класса JDatabaseQuery возвращают значение переменной $this, а следовательно, поддерживают связывание методов в цепочку. Например, следующий фрагмент кода:

```
$query = JFactory::getDbo()->getQuery(true);
$query->select('a.id, a.title');
$query->from('#__content AS a');
$query->where('a.catid = ' . (int) $catid);
```

делает то же самое, что и приведенный ниже фрагмент кода.

```
$query = JFactory::getDbo()->getQuery(true);
$query->select('a.id, a.title')->from('#__content AS a')
    ->where('a.catid = ' . (int) $catid);
```

Во втором фрагменте кода метод select() возвращает объект типа JDatabaseQueryElement, чтобы его можно было использовать для вызова метода from(). А метод from(), в свою очередь, возвращает объект, позволяющий вызвать метод where(). В базовом коде Joomla чаще всего применяется первый подход, хотя в нем можно встретить и примеры применения второго подхода.

Оператор запроса LIMIT не формируется в классе JDatabaseQuery. Но это делается при передаче запроса объекту типа JDatabase на обработку. Подробнее от этом речь пойдет далее в главе.

Группирование запросов средствами класса JDatabaseQuery

Составные запросы (т.е. запросы с оператором GROUP BY) формируются с помощью метода group(), действующего подобно оператору GROUP BY. При формировании составного запроса необходимо убедиться в том, что в списке, составляемом методом **select()**, присутствуют только те столбцы, которые используются для группирования, а также выражения с обобщающими функциями. Например, составной запрос SQL

```
SELECT a.created_by_alias, COUNT(*) AS article_count
FROM jos_content AS a
GROUP BY a.created_by_alias
```

может быть сформирован средствами класса JDatabaseQuery следующим образом:

```
$query = JFactory::getDbo()->getQuery(true);
$query->group('a.created_by_alias');
$query->from('#__content AS a');
$query->select('a.created_by_alias, COUNT(*) AS article_count');
```

Формирование запросов на ввод средствами класса JDatabaseQuery

В классе JDatabaseQuery предоставляются два альтернативных варианта для формирования запросов на ввод по команде INSERT. Один из них состоит в использовании метода set() для установки каждого столбца по отдельности, а второй — в применении методов columns() и values() для установки столбцов группой. Ниже приведен пример запроса по команде INSERT, составленный на SQL для ввода строки в таблицу соответствий jos_joompro_sub_mapping.

```
INSERT INTO jos_joompro_sub_mapping
(subscription_id, user_id, start_date, end_date)
VALUES (123, 234, '2014-01-01 12:00:00', '2014-12-31 12:00:00');
```

А вот как тот же самый запрос формируется в коде компонента подписок с помощью метода set() из класса JDatabaseQuery.

```
$query = $db->getQuery(true);
$query->insert('#__joompro_sub_mapping');
$query->set('subscription_id = ' . (int) $subID);
$query->set('user_id = ' . (int) $userID);
$query->set('start_date = ' . $db->quote($startDate));
$query->set('end_date = ' . $db->quote($endDate));
```

В качестве аргумента методу set() передается символьная строка, содержащая имя столбца, знак равенства и буквальное значение, устанавливаемое в столбце. В данном примере каждый столбец устанавливается отдельным вызовом метода set(). А несколько столбцов можно установить в одной строке кода, указав символьные строки списком через запятую. С другой стороны, то же самое можно сделать, воспользовавшись методами columns() и values(), как показано ниже.

```
$query = $db->getQuery(true);
$query->insert('#__joompro_sub_mapping');
$query->columns('subscription_id, user_id, start_date, end_date');
$query->values((int) $subID . ',' . (int) $userID . ','
    . $db->quote($startDate) . ',' . $db->quote($endDate));
```

В данном случае метод columns() служит для именования устанавливаемых столбцов. Если опустить его вызов, то в таблице будут установлены все столбцы. Затем вызывается метод values() для составления списка значений, разделяемых запятой, предназначенных для установки каждого столбца.

Формирование запросов на обновление средствами класса JDatabaseQuery

Запросы на обновление по команде UPDATE формируются с помощью методов update(), set() и where() из класса JDatabaseQuery. Ниже приведен пример запроса по команде UPDATE, составленный на SQL.

```
UPDATE jos_joompro_sub_mapping
SET start_date = '2015-01-01 12:00:00', end_date = '2015-12-31 12:00:00'
WHERE subscription_id = 123 AND user_id = 234;
```

А вот как выглядит тот же самый запрос, сформированный средствами класса JDatabaseQuery.

```
$query = $db->getQuery(true);
$query->update('#__joompro_sub_mapping');
$query->set('start_date = ' . $db->quote($startDate));
$query->set('end_date = ' . $db->quote($endDate));
$query->where('subscription_id = ' . (int) $subID);
$query->where('user_id = ' . (int) $userID);
```

В этом фрагменте кода метод set() вызывается для установки каждого обновляемого столбца, а метод where() служит для указания строк таблицы, в которых следует обновить столбцы. Как и при формировании запросов по команде INSERT, с помощью метода set() можно установить один или несколько столбцов. В последнем случае столбцы и обновляемые в них значения указываются списком через запятую.

Формирование запросов на удаление средствами класса JDatabaseQuery

Для формирования запросов на удаление по команде DELETE в классе JDatabaseQuery предусмотрены три следующих метода: delete(), from() и where(). В частности, метод delete() вызывается без аргументов и просто уведомляет свой класс о том, что формируется запрос по команде DELETE. В методе from() указывается имя таблицы, из которой удаляются строки. А в методе where() используется обычный синтаксис для составления оператора WHERE, чтобы отобрать строки для удаления.

Ниже приведен пример запроса по команде DELETE, составленный на SQL для удаления одной строки из таблицы соответствий jos_joompro_sub_mapping.

```
DELETE FROM jos_joompro_sub_mapping
WHERE subscription_id = 98 AND user_id = 76;
```

А вот как выглядит тот же самый запрос, сформированный средствами класса JDatabaseQuery.

```
$query = $db->getQuery(true);
$query->delete();
$query->from('#__joompro_sub_mapping');
$query->where('subscription_id = ' . (int) $subid);
$query->where('user_id = ' . (int) $userid);
```

Запросы на объединение в классе JDatabaseQuery

Запросы на объединение по команде UNION в классе JDatabaseQuery не поддерживаются. Как пояснялось ранее, они редко используются в приложениях Joomla.

Обработка данных по запросу

Как только запрос будет составлен, его нужно передать базе данных, выполнить, проверить ошибки и возвратить результаты, если таковые имеются. Запрос передается базе данных с помощью метода setQuery(). В частности, выполняя запросы по команде INSERT, UPDATE или DELETE, следует проверить ошибки, а выполняя запросы по команде SELECT — возвратить результаты программе.

Обработка запросов по командам INSERT, UPDATE и DELETE

Эти запросы видоизменяют базу данных, но не возвращают никаких данных программе. Ниже перечислены стадии обработки подобного рода запросов.

- Вызов метода setQuery() для передачи запроса объекту базы данных.
- Вызов метода query() для выполнения запроса.
- Проверка любых ошибок в базе данных и последующая их обработка по мере необходимости.

Рассмотрим характерный пример кода для выполнения всех стадий обработки подобных запросов. В данном примере запрос по команде UPDATE сформирован и присвоен переменной $query.

```
$db->setQuery($query);
if ($db->query()) {
    return true;
} else {
    $this->setError(JText::_('COM_JOOMPROSUBS_ADD_MAP_ROW_FAIL'));
    return false;
}
```

В первой строке приведенного выше примера кода запрос передается объекту базы данных. Если в переменной $query находится объект типа JDatabaseQuery, его метод __toString() выполняется автоматически. Этот метод формирует запрос, используя синтаксис, требующийся для конкретной применяемой базы данных.

Метод query() выполняет запрос и возвращает логическое значение true, если выполнение запроса завершится успешно. Следовательно, возвращаемое им значение может быть использовано для проверки ошибок, которые могли возникнуть при выполнении запроса в базе данных. Если таковые обнаружатся, то обрабатывается состояние ошибки. Следует заметить, что для получения дополнительных сведений об ошибке в базе данных можно воспользоваться методами $db->getErrorNum() и $db->getErrorMsg().

Обработка запросов по команде SELECT

Ниже перечислены стадии обработки запросов по команде SELECT.

- Вызов метода **setQuery()** для передачи запроса объекту базы данных. Этот метод служит также для дополнительного ограничения количества строк, возвращаемых из таблицы по запросу.

- Вызов одного из методов загрузки для обработки запроса и возврата результатов в переменной языка PHP.
- Проверка любых ошибок в базе данных и последующая их обработка по мере необходимости.

Рассмотрим характерный пример кода для выполнения всех стадий обработки подобных запросов. Соответствующий код приведен ниже.

```
$db->setQuery($query, $limitstart, $limit);
$data = $db->loadObjectList();
if ($db->getErrorNum()) {
    $this->setError(JText::_('COM_MYCOMPONENT_ERROR_MSG'));
    return false;
} else {
    return $data;
}
```

Как и прежде, в первой строке этого примера кода сформированный запрос передается объекту базы данных. Но в данном случае методу `setQuery()` в качестве дополнительных аргументов передаются переменные `$limitstart` и `$limit`. Значения этих переменных предназначаются для поддержки режима разбиения на страницы, в противном случае — для ограничения набора данных, возвращаемых по запросу, фиксированным количеством строк.

В данном примере с помощью переменной `$limitstart` базе данных сообщается количество строк, пропускаемых в данных, возвращаемых по запросу, а с помощью переменной `$limit` — максимальное количество возвращаемых строк. Так, если значение переменной `limitstart` равно **0**, а значение переменной `$limit` — **15**, то по запросу будет возвращено до 15 первых строк из набора данных. Если же значение переменной `limitstart` равно **15**, а значение переменной `$limit` — **5**, то первые 15 строк пропускаются и возвращаются 5 последующих строк (с 16-й по 20-ю включительно), при условии, что набор данных по запросу содержит такое количество строк из таблицы. Если же значение переменной `$limit` оказывается больше, чем количество строк в этом наборе данных, то по запросу не будет возвращено ни одной строки таблицы.

Во второй строке рассматриваемого здесь кода вызывается метод `loadObjectList()` для возврата результатов обработки запроса в виде массива объектов, представляющих строки таблицы. Различные варианты возврата результатов после обработки запроса по команде `SELECT` будут рассмотрены несколько ниже.

И в третьей строке кода вызывается метод `getErrorNum()` для проверки ошибок в базе данных. Если таковые обнаружатся, составляется сообщение об ошибке и возвращается логическое значение `false`. В противном случае результаты обработки запроса возвращаются в переменной `$data`. Потребность в проверке ошибок, возникающих в базе данных, и реагировании на них зависит от конкретной ситуации. В одних случаях обработка данных продолжается как обычно, а в других — пользователь уведомляется об ошибке. Когда на платформе Joomla появятся новые функциональные возможности для регистрации ошибок, некритичные ошибки в базе данных будут регистрироваться в системном журнале, чтобы администратор веб-сайта мог просмотреть его и исправить эти ошибки, не создавая неблагоприятного впечатления пользователей от веб-сайта.

Возврат результатов обработки запроса

После обработки запросов по команде SELECT данные возвращаются из таблицы в строках и столбцах. В зависимости от конкретного запроса может быть возвращено одно значение, одна строка или же несколько строк. В классе JDatabase предоставляется целый ряд методов для выборки данных из запроса по команде SELECT. В табл. 11.2 перечислены наиболее употребительные методы, возвращающие результаты обработки запросов по команде SELECT.

Таблица 11.2. Методы из класса JDatabase, возвращающие данные по запросу на выборку

Метод из класса JDatabase	Тип возвращаемых данных	Описание
loadObject()	Стандартный объект	Возвращает первую строку по запросу
loadObjectList()	Индексированный массив стандартных объектов	Возвращает все строки по запросу
loadResult()	Смешанный	Возвращает первый столбец из первой строки
loadColumn()	Массив	Возвращает по запросу массив, состоящий из одного столбца
loadAssoc()	Ассоциативный массив	Возвращает первую строку по запросу
loadAssocList()	Индексированный массив ассоциативных массивов	Возвращает все строки по запросу
loadRow()	Индексированный массив	Возвращает по запросу первую строку в виде индексированного массива
loadRowList()	Индексированный массив индексированных массивов	Возвращает все строки по запросу

Для загрузки результатов обработки запросов в Joomla чаще всего применяются методы loadObjectList() и loadResult(). В частности, метод loadObjectList() вызывается в том случае, если по запросу из таблицы может быть возвращено несколько строк и столбцов. Возвращаемым типом в данном случае оказывается массив, состоящий из 0–n стандартных объектов. Поля стандартных объектов содержат имена столбцов и значения из этих столбцов. Для того чтобы воспользоваться в коде PHP значением, возвращаемым данным методом, можно организовать цикл foreach, как показано в приведенном ниже примере.

```
$data = $dbo->loadObjectList();
foreach ($data as $row) {
  $title = $row->get('title');
  $id = $row->get('id');
. . .
```

В этом примере кода переменная $row содержит стандартный объект, представляющий одну строку таблицы из набора данных, возвращаемых по запросу. Для извлечения

из этой строки отдельных столбцов с данными по имени столбца, указанного в запросе, можно вызвать метод `get()` из стандартного объекта. Метод `loadResult()` обычно применяется в тех случаях, когда по запросу предполагается возвратить только одно значение (одну строку или столбец), например, название единственной категории. И в этом случае возвращаемое значение просто присваивается единственной переменной.

Метод `loadObject()` действует аналогично методу `loadObjectList()`, за исключением того, что он возвращает по запросу лишь первую строку из таблицы. Он применяется в тех случаях, когда по запросу предполагается получить только одну строку таблицы или же когда нужна лишь первая ее строка. В итоге возвращается стандартный объект, поля которого опять же содержат имена столбцов и значения из этих столбцов.

Метод `loadAssocList()` очень похож на метод `loadObjectList()`. Он возвращает массив, каждый элемент которого является ассоциативным массивом, содержащим данные из одной строки таблицы. В приведенном ниже примере кода демонстрируется, каким образом обрабатываются результаты, возвращаемые данным методом.

```
$data = $dbo->loadAssocList();
foreach ($data as $row) {
    $title = $row['title'];
    $id = $row['id'];
. . .
```

Этот код похож на приведенный выше код обработки результатов, возвращаемых методом `loadAssocList()`, за исключением того, что доступ к каждой строке осуществляется в нем как к ассоциативному массиву, а не как к стандартному объекту.

Метод `loadAssoc()` подобен методу `loadObject()`. Оба метода возвращают по запросу только первую строку таблицы. Но метод `loadAssoc()` возвращает ее в виде ассоциативного массива.

Метод `loadRowList()` похож на метод `loadAssocList()`, за исключением того, что он возвращает индексированный, а не ассоциативный массив. В приведенном ниже примере кода демонстрируется, каким образом обрабатываются результаты, возвращаемые данным методом.

```
$data = $dbo->loadRowList();
foreach ($data as $row) {
    $first_column = $row[0];
    $second_column = $row[1];
. . .
```

Метод `loadColumn()` позволяет обрабатывать данные из одного столбца, представленные в виде индексированного массива. Он обычно применяется в тех случаях, когда запрос содержит несколько строк по одному столбцу в каждой строке. Этот метод возвращает индексированный массив, где каждый элемент содержит данные из столбца, относящегося к строке с заданным значением индекса. Так, если по запросу из таблицы возвращаются три строки, первый элемент массива `$data[0]` будет содержать значение из первой строки, второй элемент массива `$data[1]` — значение из второй строки, а третий элемент массива `$data[2]` — значение из третьей строки таблицы.

Резюме

В этой главе были представлены основы работы с базами данных SQL, включая построение структуры таблиц базы данных и составление запросов SQL для обновления и доступа к информации, хранящейся в базе данных. Кроме того, в ней были рассмотрены особенности составления запросов в коде приложений Joomla, а также выборки и обработки данных по этим запросам.

12

Применение JavaScript и MooTools в Joomla!

Язык JavaScript обеспечивает улучшение взаимодействия пользователя с приложениями и постепенно становится очень важной частью Joomla!. Кроме того, JavaScript является ключевым компонентом технологии AJAX (асинхронного сочетания JavaScript и XML). А MooTools — это интегрированная среда JavaScript, включенная в состав Joomla. Благодаря MooTools упрощается процесс создания и сопровождения функциональных средств JavaScript, надежно действующих в разнотипных браузерах.

В этой главе сначала рассматриваются основные понятия, связанные с JavaScript и MooTools и их интеграцией в Joomla. Затем обсуждаются конкретные средства JavaScript, встроенные в Joomla, а также их применение в программах. И в завершение главы применение JavaScript будет продемонстрировано на примере создания обновляемого средствами AJAX варианта страницы веб-сайта на Joomla.

JavaScript представляет собой полноценный язык программирования с присущими ему особенностями и недостатками. А MooTools является эффективной интегрированной средой со многими базовыми средствами и еще большим числом расширений. Вопросы освоения JavaScript и MooTools выходят за рамки этой книги. Правда, конструктивные особенности Joomla позволяют выгодно воспользоваться многими средствами JavaScript и MooTools, не особенно вдаваясь в изучение языка JavaScript.

Общее представление о JavaScript

JavaScript — это язык событийно-ориентированного программирования, действующий в браузере. Как и PHP, JavaScript является интерпретируемым языком, а следовательно, сценарии на нем выполняются непосредственно, т.е. без предварительной компиляции. Но, в отличие от языка PHP, программы на котором работают

непосредственно на веб-сервере, сценарии на JavaScript выполняются на клиентской машине, например ПК пользователя. Всякий современный браузер содержит интерпретатор JavaScript, способный выполнять код JavaScript, включаемый в состав веб-страницы. А поскольку код JavaScript выполняется локально на клиентской машине, то он способен мгновенно реагировать на события, не требуя установления связи с веб-сервером. Именно по этой причине программы, написанные на JavaScript, действуют подобно настольным прикладным программам на C, Java или Basic, а не типичным веб-ориентированным приложениям.

Принцип действия JavaScript

В Joomla код JavaScript действует наряду с кодом PHP. Он не подменяет собой код или функции PHP, а только улучшает их. Благодаря JavaScript веб-ориентированные приложения Joomla становятся более интерактивными, чем в том случае, если бы они были разработаны только на PHP и HTML.

Сценарий JavaScript действует на клиентской стороне, динамически видоизменяя HTML-документ, в том числе добавляя, изменяя или удаляя элементы разметки и атрибуты HTML. Такой сценарий реагирует на события, связанные с действиями пользователя или браузера. К числу типичных событий относятся следующие:

- Перемещение курсора мыши к отдельному элементу HTML-документа или в сторону от него.
- Выполнение щелчка на каком-нибудь элементе HTML-документа.
- Изменение значения в форме.
- Предъявление формы.
- Завершение процесса загрузки HTML-документа.

Так, если курсор мыши наводится на отдельный элемент, расположенный на странице, функция JavaScript может видоизменить HTML-документ в рабочей области памяти браузера, чтобы отобразить всплывающую подсказку. При этом функция JavaScript добавляет элементы разметки всплывающей подсказки в HTML-документ, и в результате она появляется на странице. Когда же курсор мыши отводится от данного элемента, HTML-документ снова видоизменяется, чтобы удалить из него элементы разметки всплывающей подсказки, после чего она исчезает. И все это происходит мгновенно в рабочей области памяти браузера, не требуя установления связи с веб-сервером.

JavaScript можно также использовать для составления запросов AJAX. Это запросы веб-сервера, аналогичные обычным запросам на загрузку страниц, но отличаются они тем, что выполняются в фоновом режиме, не прерывая работу программы. В то время как пользователь продолжает работать с прикладной программой, запрос направляется на веб-сервер, а затем от него поступает ответ.

Обе особенности JavaScript (мгновенное реагирование на действия пользователя и установление связи с веб-сервером в фоновом режиме) позволяют разработчикам создавать у пользователей более благоприятное и полное впечатление от взаимодействия с их веб-приложениями, чем это могло бы быть в противном случае.

Программы на JavaScript, чаще всего называемые сценариями, включаются в состав HTML-страниц с помощью элемента разметки `script`. Наиболее предпочтительный

способ размещения кода JavaScript на HTML-странице состоит в том, чтобы ввести один или более элемент разметки `script` в элемент разметки `head` этой страницы. В Joomla это обычно делается с помощью методов поведения из класса `JHTML`. Например, в строке кода

```
Html::_('behavior.tooltip');
```

вводится код JavaScript, требующийся для отображения всплывающих подсказок на HTML-странице.

Общее представление о MooTools

MooTools — это интегрированная среда JavaScript, обладающая двумя главными преимуществами. Во-первых, она предоставляет немало встроенных функциональных средств, в том числе встроенные функции для создания таких распространенных элементов пользовательского интерфейса, как слайдеры, всплывающие подсказки, развертываемые иерархические деревья и пр. Эти средства доступны в Joomla через простой прикладной программный интерфейс (API) и не требуют написания базового кода JavaScript.

Во-вторых, эта интегрированная среда обеспечивает совместимость с браузерами. Несмотря на то что все современные браузеры способны выполнять сценарии JavaScript, имеется немало мелких отличий в том, как эти сценарии действуют в различных браузерах, что, безусловно, затрудняет написание кода JavaScript, надежного работающего во всех средах. Поэтому разработчики интегрированной среды MooTools постарались обеспечить совместимость ее средств и согласованность их работы во всех поддерживаемых браузерах. И там, где возникают трудности совместимости с отдельными браузерами, в данную интегрированную среду встроен код для преодоления этих трудностей. Разработчики MooTools хорошо разбираются в тонкостях работы интерпретаторов JavaScript в разных браузерах, а это означает, что пользователи MooTools избавлены от необходимости самостоятельно разбираться во всех этих тонкостях. Код, который они пишут, используя интерфейс API интегрированной среды MooTools, будет корректно выполняться во всех поддерживаемых браузерах.

Таким образом, интегрированная среда, подобная MooTools, позволяет сэкономить время, затрачиваемое на разработку приложений, поскольку разработчики могут выгодно воспользоваться встроенными в нее функциональными средствами. Она также позволяет сэкономить время и труд, затрачиваемые на сопровождение приложений, избавляя от необходимости тестировать и отлаживать код JavaScript в связи с его применением в различных браузерах.

Имеется также ряд других распространенных интегрированных сред JavaScript, в том числе jQuery, Dojo и Prototype. Каждой из них присущи свои достоинства и недостатки, и у каждой имеются свои сторонники. Начиная с версии 1.5 в проекте Joomla было решено использовать интегрированную среду MooTools, поскольку она обладает следующими основными преимуществами.

- **Облегченность и модульность.** Этой средой можно пользоваться по частям, не замедляя заметно загрузку страниц.
- **Объектно-ориентированное программирование.** Как и в Joomla, в MooTools применяется объектно-ориентированное программирование (ООП), и поэтому эта среда вполне подходит для проектирования программного обеспечения.

- **Естественная среда для JavaScript.** Интегрированная среда MooTools расширяет JavaScript, не изменяя порядок написания кода JavaScript. Освоение MooTools является естественным продолжением изучения языка JavaScript.

Перечисленные выше преимущества MooTools сыграли решающую роль, когда решение принималось руководителями проекта Joomla в 2007 году. И, по мнению Ведущей производственной группы проекта Joomla, эти преимущества остаются актуальными до сих пор. В настоящее время MooTools представляет собой зрелый общедоступный проект, который продолжает развиваться и совершенствоваться, и поэтому в ближайшем будущем переход Joomla на другую интегрированную среду JavaScript не планируется.

Как поясняется далее в главе, разработчики могут включать и другие интегрированные среды JavaScript в шаблоны и расширения Joomla. Изменения, внесенные в MooTools и Joomla, начиная с версии 1.6, позволяют сделать это относительно просто. Но поскольку среда MooTools уже включена в каждую установку Joomla и довольно тесно интегрирована с платформой Joomla и системой управления информационным наполнением (CMS), разработчикам в Joomla рекомендуется пользоваться MooTools.

Порядок применения JavaScript и MooTools в Joomla

Joomla следует стандартам на ненавязчивый JavaScript. Это означает следующее.

- Код JavaScript отделяется от разметки в коде HTML и выбирает элементы этой разметки для изменения, исходя из их атрибутов, например `id` или `class`.
- Большинство страниц в Joomla разработано с учетом постепенного снижения функциональных возможностей. Так, если функции JavaScript запрещены в браузере, у пользователя должна быть по-прежнему возможность для доступа ко всем функциональным возможностям веб-сайта, хотя и, вероятнее всего, с менее изящным пользовательским интерфейсом.
- Страницы и функциональные возможности веб-сайта на Joomla должны быть доступны для пользователей со слабым зрением или другими физическими ограничениями. Улучшение впечатления пользователей от взаимодействия с веб-сайтом с помощью JavaScript не должно, например, мешать одним людям со слабым зрением пользоваться программами чтения с экрана, а другим — прикладными программами, рассчитанными на ввод данных только с клавиатуры, но без применения мыши.

Реализация JavaScript в Joomla, как правило, включает в себя вызов одного из методов поведения класса `JHTML`, например `JHtml::_('behavior.tooltip')`. Методы поведения из класса `JHTML` загружают требующийся код JavaScript в заголовок HTML-документа, чтобы сделать его доступным для выполнения, если он будет вызван. Этого оказывается достаточно для реализации некоторых свойств, например, поддержания страниц в активном состоянии. А для реализации других свойств, например всплывающих подсказок, приходится проверять наличие подходящей разметки в коде HTML для вызова соответствующего поведения JavaScript.

Встроенные средства JavaScript

В табл. 12.1 перечислены стандартные виды поведения JavaScript, доступные в версии Joomla 2.5.

Таблица 12.1. Встроенные в Joomla виды поведения JavaScript

Поведение	Описание	Файл JavaScript
`calendar`	Предоставляет всплывающий календарь для ввода дат, выбираемых щелчком кнопкой мыши	`calendar.js`
`caption`	Создает надписи под изображениями при загрузке страницы	`caption.js`
`colorpicker`	Предоставляет всплывающий селектор цвета	`mooRainbow.js`
`formvalidation`	Обеспечивает проверку достоверности форм типа `JForm`	`validate.js`
`framework`	Загружает базовые средства интегрированной среды MooTools, когда они требуются в специальных сценариях JavaScript; характерным тому примером служит элемент управления, устанавливающий сразу все флажки	`mootools.js`
`highlighter`	Выделяет слова фоновым цветом	`highlighter.js`
`keepalive`	Сохраняет активным сеанс работы, например, при выполнении длительной задачи редактирования	Отсутствует (используется файл `mootools.js`)
`modal`	Предоставляет всплывающее режимное окно, не требуя обновления страницы	`modal.js`
`multiselect`	Позволяет установить ряд флажков на экране одновременным нажатием клавиши <Shift> и щелчком левой кнопкой мыши	`multiselect.js`
`noframes`	Предотвращает загрузку страницы во встраиваемый фрейм (например, с целью воспрепятствовать захвату сайта)	Отсутствует (используется файл `mootools.js`)
`switcher`	Предоставляет похожие на вкладки панели на экране (например, на экране Global Configuration)	`switcher.js`
`tooltip`	Предоставляет всплывающие подсказки при наведении курсора мыши на элемент страницы	Отсутствует (используется файл `mootools.js`)
`tree`	Предоставляет элемент управления развертываемым иерархическим деревом	`mootools.js`
`uploader`	Предоставляет загрузчик файлов на веб-сайт (например, на экране Media Manager)	`uploader.js`

Файлы JavaScript находятся в папке `media/system/js`. Следует заметить, что в пакет Joomla входят по две копии каждого файла JavaScript. Все файлы, перечисленные в табл. 12.1, фактически используются в Joomla. Они сделаны компактными с целью свести к минимуму время их загрузки. В частности, из них удалены все пробелы и комментарии. А вторая копия каждого файла со словом `"-uncompressed"` в его имени содержит полный, отформатированный обычным способом исходный текст с комментариями и предназначена для отладки, чтения и редактирования кода.

Упомянутые выше виды поведения JavaScript рассматриваются далее по отдельности.

Поведение calendar

Этот вид поведения создает всплывающий календарь, дающий пользователю возможность ввести дату с помощью визуального календарного виджета. На рис. 12.1 приведен пример такого календарного виджета из экранной формы редактирования статьи.

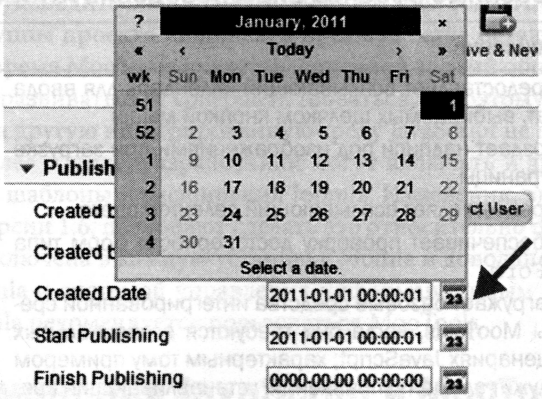


Рис. 12.1. Всплывающий календарь

Такой виджет может быть введен на странице одним из двух способов. Самый простой способ состоит в использовании поля объекта типа `JForm`, в атрибуте `type` которого задан календарь. Например, выделенный полужирным код в приведенном ниже фрагменте разметки из файла `administrator/components/com_content/models/forms/article.xml` предписывает объекту типа `JForm` воспроизвести данное поле с всплывающим календарем.

```
<field name="created" type="calendar"
label="COM_CONTENT_FIELD_CREATED_LABEL"
  description="COM_CONTENT_FIELD_CREATED_DESC" class="inputbox" size="22"
  format="%Y-%m-%d %H:%M:%S" filter="user_utc" />
```

Напомним, что атрибут `type` определяет, какой именно класс типа `JFormField` используется для воспроизведения поля. В данном случае из класса `JFormFieldCalendar` вызывается метод `getInput()`. Ниже приведена последняя строка кода из этого метода.

```
return JHtml::_('calendar', $this->value, $this->name, $this->id, $format,
↳$attributes);
```

В этой строке вызывается метод `JHtml::calendar()` для воспроизведения поля календаря.

Если же календарь требуется использовать за пределами формы типа `JForm`, то метод `JHtml::calendar()` можно вызвать непосредственно. В таком случае ему в качестве аргументов передаются дата, имя поля, идентификатор поля, формат даты, а также массив с любыми дополнительными атрибутами, которые требуются для разметки поля в формате HTML. Например, в приведенном ниже фрагменте кода из файла `administrator/components/com_banners/views/tracks/tmpl/default.php` формируется начальная дата для фильтра дат на экране Banner Manager: Tracks (Диспетчер баннеров: Каналы отслеживания баннеров).

```
<?php echo JHtml::_('calendar',$this->state->get('filter.begin'),
'filter_begin','filter_begin','%Y-%m-%d' ,
array('size'=>10,'onchange'=>"this.form.fireEvent(
'submit');this.form.submit()"));?>
```

В данном примере кода указываемый в качестве аргумента массив содержит два атрибута разметки в формате HTML: `size`, задающий размер поля, а также `onchange`, определяющий условие предъявления формы всякий раз, когда в поле `filter_begin` вносятся изменения. Таким образом, страница перезагружается с нужными элементами, когда в поле `filter_begin` происходят изменения.

Каким бы способом ни создавался календарь, вводить в код вызов метода поведения `JHtml::_('behavior.calendar')` совсем не обязательно. Ведь это делается автоматически в методе `JHtml::calendar()`.

Поведение `caption`

Этот вид поведения автоматически добавляет текст надписи под изображением. Для того чтобы воспользоваться им, в элемент разметки `img` придется ввести два атрибута:

- `class="caption"`: используется в качестве селектора для функции JavaScript.
- `title="<desired caption title>"`: текст для отображения надписи.

Данное поведение JavaScript автоматически выбирает любые элементы разметки `img` с классом `"caption"` и формирует выравниваемую по центру надпись под изображением, используя значение, устанавливаемое в атрибуте `title` для текста надписи. Поведение `caption` служит характерным примером ненавязчивости JavaScript. Для вызова этого поведения требуется лишь наличие упомянутых выше атрибутов в коде разметки. Если же функции JavaScript запрещены, это поведение благополучно перестает действовать. Единственным последствием этого оказывается отсутствие надписи под изображением.

Поведение `colorpicker`

Этот вид поведения является новым для версии Joomla 2.5. Такое поведение создает всплывающий элемент управления, дающий пользователю возможность выбрать нужное значение цвета. На рис. 12.2 приведен пример всплывающего селектора цвета, определяемого поведением `colorpicker`. Цвет выбирается визуально путем перемещения ползунка по палитре цветов. А результатом выбора оказывается шестнадцатеричное значение цвета.

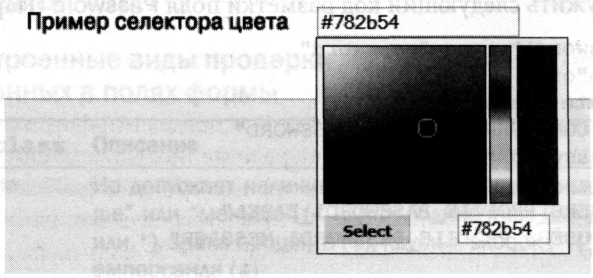


Рис. 12.2. Пример всплывающего селектора цвета, определяемого поведением `colorpicker`

Элемент управления `colorpicker` нетрудно создать средствами класса `JForm`, указав значение `"color"` в атрибуте `type`. А для создания элемента управления `colorpicker` без класса `JForm` достаточно выполнить следующие действия.

1. Ввести вызов метода поведения `JHtml::_('behavior.colorpicker');` в код, чтобы загрузить нужный файл JavaScript.
2. Создать элемент разметки `input` с атрибутами `type="text"` и `class="input-colorpicker"` (выбираемый класс CSS).

Проверка достоверности формы

Поведение `formvalidation` позволяет проверить достоверность данных в полях форм средствами JavaScript. Проверка достоверности средствами JavaScript обладает тем преимуществом, что дает возможность немедленно уведомить пользователя, не дожидаясь предъявления формы, если он введет недостоверные данные в поле формы. На рис. 12.3 приведен пример действия поведения для проверки достоверности формы на экране Edit Your Profile (Правка вашего профиля) в пользовательской части веб-сайта.

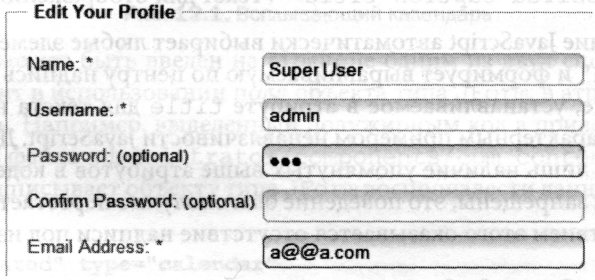


Рис. 12.3. Пример проверки достоверности формы

В данном примере в полях формы введены неверный пароль и адрес электронной почты. Оба поля выделяются, как только в них происходят изменения. И для того чтобы увидеть ошибки ввода данных, совсем не обязательно щелкать на кнопке Submit.

Рассмотрим подробнее действие данного вида поведения. В XML-файле разметки упомянутой выше формы типа `JForm` (`components/com_users/models/forms/profile.xml`) можно обнаружить следующий код разметки поля Password (Пароль):

```
<field name="password1" type="password"
    autocomplete="off"
    class="validate-password"
    description="COM_USERS_DESIRED_PASSWORD"
        field="password2"
    filter="raw"
    label="COM_USERS_PROFILE_PASSWORD1_LABEL"
    message="COM_USERS_PROFILE_PASSWORD1_MESSAGE"
    size="30"
    validate="equals"
/>
```

В атрибуте `class` установлено значение `"validate-password"` (выделено выше полужирным), а это означает, что при вводе пароля в данном поле автоматически вызывается проверка достоверности средствами JavaScript. Ниже приведен код разметки поля Email Address (Адрес электронной почты) из того же самого XML-файла.

```
<field name="email1" type="email"
    description="COM_USERS_PROFILE_EMAIL1_DESC"
    filter="string"
    label="COM_USERS_PROFILE_EMAIL1_LABEL"
    message="COM_USERS_PROFILE_EMAIL1_MESSAGE"
    required="true"
    size="30"
    unique="true"
    validate="email"
/>
```

В атрибуте `type` установлено значение `"email"` (выделено выше полужирным), а это означает, что для воспроизведения данного поля в форме типа `JForm` должен быть использован класс `JFormFieldEMail`. В файле компоновки `libraries/joomla/form/fields/email.php` устанавливается значение `"validate-email"` атрибута `class` разметки данного поля, и благодаря этому автоматически запускается проверка достоверности адреса электронной почты средствами JavaScript.

То же самое поведение может быть использовано и для проверки ввода данных в поле, которое должно быть обязательно заполнено. Для этого в атрибуте `required` устанавливается логическое значение `true`. Это можно сделать в файле разметки полей формы типа `JForm` или же просто ввести данный атрибут в элемент разметки `input` соответствующего поля. Все эти виды проверки достоверности выполняются в файле `media/system/js/validate.js` сценария JavaScript. Для того чтобы поведение `formvalidation` начало действовать, достаточно выполнить следующие действия.

- Вызвать метод поведения `JHtml::_('behavior.formvalidation')`, чтобы загрузить соответствующий сценарий JavaScript в заголовок страницы.
- Ввести атрибут `class="form-validate"` в элемент разметки `form`.
- Ввести атрибут `required="true"` или атрибут `class` с одним из значений, указанных в табл. 12.2, в элемент разметки `field`. Это делается в XML-файле для соответствующего поля формы типа `JForm`. (Как отмечалось выше, атрибут `class` вводится автоматически, если в разметке поля формы типа `JForm` задан атрибут `type` со значением `"email"`.)

Таблица 12.2. Встроенные виды проверки достоверности данных в полях формы

Значение атрибута `class`	Описание
`validate-username`	Не допускает наличия в поле следующих символов: знаков "больше" или "меньше" (< или >), одиночных или двойных кавычек (" или '), знака процента (%), круглых скобок (" (" или ") ") или знака амперсанда (&)
`validate-password`	Допускает не меньше **4**, но не больше **196** символов в пароле, который не должен начинаться с символа пробела

Окончание табл. 12.2

Значение атрибута `class`	Описание
`validate-numeric`	Допускает наличие только цифр от **0** до **9**, знака "минус" (-), запятой (,) или точки (.)
`validate-email`	Допускает наличие символьной строки в формате `xxx@yyy.zzz`, где подстроки `xxx` и `yyy` должны иметь длину не меньше одного символа, а подстрока `zzz` — от двух до четырех символов

Следует иметь в виду, что рассматриваемое здесь поведение JavaScript действует отдельно от проверки достоверности, выполняемой средствами определенного класса `JFormRule` и запускаемой с помощью атрибута `validate` в разметке полей формы типа `JForm`. Как пояснялось в главе 6, если в разметку поля формы типа `JForm` вводится атрибут `validate`, то из соответствующего класса `JFormRule` автоматически вызывается метод `test()`. Например, для проверки достоверности данных в поле ввода адреса электронной почты устанавливается значение `"email"` в атрибуте `validate`, что влечет за собой вызов метода `test()` из класса `JFormRuleEMail`. Выполнение этого метода начинается после щелчка на кнопке Submit. Следует также иметь в виду, что такая проверка отличается от обычной фильтрации вводимых данных. Ведь фильтрация, по существу, изменяет значения в полях во время сохранения формы, чтобы удалить из них потенциально опасные символы.



Как пояснялось ранее в этой книге, ни один из видов проверки достоверности данных не защищает полностью от злоумышленников. Рассматриваемый здесь вид проверки достоверности имеет большое значение для предоставления пользователям удобного интерфейса, но не избавляет от дополнительной проверки достоверности и фильтрации данных после предъявления формы, чтобы надежно защитить веб-сайт от внесения злонамеренного кода.

Запрет действий по предъявлению или сохранению формы

Как пояснялось ранее, проверка достоверности средствами JavaScript действует локально в веб-браузере. Она не требует от пользователя щелкать на кнопке Submit или Save, чтобы проверить достоверность данных, введенных в полях формы. Если некоторые данные, введенные в форме, оказываются недействительными в результате проверки их достоверности средствами JavaScript, то может возникнуть потребность запретить действие по предъявлению формы до тех пор, пока во всех ее полях не окажутся действительные данные.

Если же попытаться предъявить форму с недействительными данными, то из веб-сервера должны быть получены сообщения об ошибках проверки достоверности, и тогда форма не будет принята. Следовательно, предъявлять форму с недействительными данными нет никакого смысла, да и пользователю будет не совсем удобно получать сообщения только об одном виде ошибок проверки достоверности вместо того, чтобы получать сообщения об одних ошибка до предъявления формы, а о других — после него.

В базовых программах Joomla функция предъявления формы, как правило, запрещается, когда обнаруживаются ошибки проверки достоверности. И делается это двумя разными способами.

- Если в компоновке формы имеется элемент разметки `button` с атрибутами `type="submit"` и `class="validate"`, то проверка достоверности данных будет автоматически выполнена во всей форме, и действие этой кнопки по предъявлению формы будет запрещено, если в данной форме окажутся какие-нибудь недействительные данные.
- Если же в компоновке формы отсутствует кнопка для ее предъявления, то можно создать функцию JavaScript для перехвата действия по предъявлению формы, предъявив ее только в том случае, если все данные в ней окажутся действительными. Именно так и происходит на экранах из административной части веб-сайта, где имеются панели инструментов с кнопками для выполнения различных задач.

Пример реализации второго способа можно обнаружить в файле компоновки `administrator/components/com_users/views/user/tmpl/edit.php` экрана **User Manager: Edit** (Диспетчер пользователей: Правка). В частности, в этом файле имеется следующий фрагмент кода:

```
<script type="text/javascript">
    Joomla.submitbutton = function(task)
    {
      if (task == 'user.cancel' ||
↳document.formvalidator.isValid(document.id('user-form'))) {
        Joomla.submitform(task, document.getElementById('user-form'));
    }
  }
</script>
```

В этом фрагменте кода создается функция JavaScript под названием `Joomla.submitbutton`. Эта функция вызывается, когда пользователь щелкает кнопкой мыши на пиктограмме любой кнопки, доступной на панели инструментов, например **Save**, **Cancel** или **Save & Close**. Если выбрана задача `user.cancel` или форма окажется достоверной, она предъявляется при вызове функции `Joomla.submitform`, определенной в файле `media/system/js/core.js`. В противном случае ничего не происходит, в следовательно, форма не предъявляется. Таким образом, пользователь должен исправить любые ошибки, допущенные при вводе данных в форме, прежде чем ее можно будет предъявить.

Это еще один характерный пример ненавязчивости JavaScript. Приложение не полагается на проверку достоверности данных в полях формы только средствами JavaScript. Для этой цели в нем имеются другие методы, в том числе из классов типа `JFormRule` и `JTable`. А функции JavaScript лишь улучшают пользовательский интерфейс, обеспечивая немедленную реакцию на ввод пользователем недействительных данных. Если же функции JavaScript запрещены, форма будет действовать по-прежнему, но с менее удобным для пользователя интерфейсом.

Поведение framework

Этот вид поведения просто загружает интегрированную среду MooTools из дескриптора `script` в заголовке страницы. Оно служит для ввода специального кода JavaScript, опирающегося на MooTools. Это поведение никак не связано ни с одной из функций JavaScript, а в коде оно вызывается следующим образом:

```
JHtml::_('behavior.framework');
```

Поведение highlighter

Этот вид поведения позволяет легко выделять выбранные слова на странице. Оно используется в компоненте Smart Seach (Интеллектуальный поиск) для выделения критериев поиска, обнаруженных в тексте статьи, как показано на рис. 12.4.

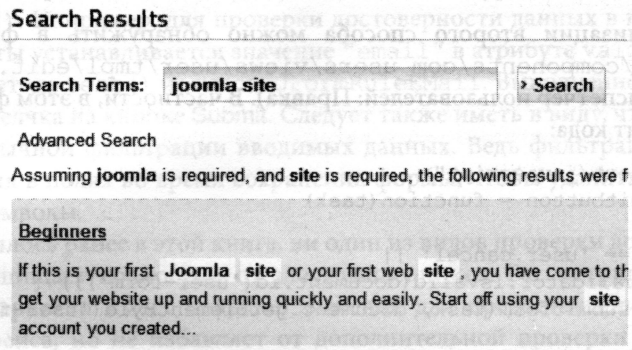


Рис. 12.4. Пример действия поведения `highlighter`

Для того чтобы воспользоваться поведением `highlighter`, выполните следующие действия.

1. Введите в исходный код вызов метода поведения `JHtml::_('behavior.highlighter', $wordArray)`, где `$wordArray` — массив слов, которые требуется выделить в тексте.
2. Введите код `<br id="highlighter-start" />` в разметку HTML-документа непосредственно перед выделяемым разделом этого документа.
3. Введите код `<br id="highlighter-end" />` в разметку HTML-документа сразу же после выделяемого раздела этого документа.

Поведение keepalive

Это поведение позволяет Joomla сохранять активным сеанс работы неопределенно долго. Как правило, сеанс работы пользователя на веб-сайте Joomla истекает через установленный промежуток времени (по умолчанию через 15 минут). Это делается из соображений безопасности. Так, если пользователь войдет в уязвимую часть веб-сайта и оставит свой компьютер, не выйдя из этой части, сеанс его работы следует ограничить по времени, чтобы не дать возможности посторонним лицам получить доступ к веб-сайту.

Но в некоторых случаях ограничение по времени может вызвать определенные затруднения. Допустим, пользователь редактирует длинную статью. Нетрудно представить, что такой сеанс работы может продолжаться больше 15 минут без сохранения промежуточных результатов. Если сеанс работы истечет за это время, пользователь может потерять результаты своих трудов. Более того, он обнаружит это лишь после щелчка на кнопке Save, когда будет слишком поздно! Во избежание этого можно вызвать поведение `keepalive`. Для этого достаточно ввести лишь одну, приведенную ниже строку кода.

```
JHtml::_('behavior.keepalive');
```

В этой строке кода вызывается встроенный в MooTools метод поведения, автоматически делающий запрос AJAX на веб-сервер за минуту до истечения сеанса работы. Этот запрос выполняется в фоновом режиме, вынуждая веб-сервер сбросить установленное время истечения сеанса работы. В итоге сеанс работы будет продолжаться неопределенно долго. Данное поведение используется на экранах редактирования многих компонентов Joomla.

Поведение `modal`

Этот вид поведения используется в том случае, если на экране требуется отобразить режимное всплывающее окно. Ниже приведены два примера применения этого вида поведения в базовом коде Joomla.

- Выбор кнопки Options на панели инструментов экрана User Manager и других экранов, чтобы отредактировать параметры настройки конфигурации на уровне компонента.
- Выбор кнопки Select/Change, чтобы подобрать статью для пункта меню Single Article (Единственная статья).

В подобных случаях нужно сначала сделать запрос веб-сервера, чтобы получить данные, а затем отобразить их. Но при этом нежелательно терять текущий контекст. Ведь после закрытия режимного окна необходимо продолжить работу с того места на экране, где она была прервана открытием режимного окна.

Поведение `modal` дает возможность открыть режимное окно в текущем документе, поработать в нем и закрыть его по завершении работы. После закрытия режимного окна происходит возврат в родительское окно. И до тех пор, пока режимное окно открыто, работать в родительском окне нельзя. Как правило, режимное окно открывается в элементе разметки `iframe`. Это дает возможность открыть подчиненный документ внутри родительского. Для применения поведения `modal` в Joomla необходимо выполнить следующие действия.

- Ввести в исходный код вызов метода поведения `JHthml::_('behavior.modal')`, чтобы вставить соответствующий сценарий JavaScript в заголовок страницы. Селектор класса CSS можно передать методу данного поведения в качестве второго дополнительного аргумента. По умолчанию этот аргумент принимает значение `a.modal`, означающее выбор элемента привязки `a` с атрибутом `class="modal"`.
- Создать элемент разметки `anchor` с атрибутом `class="modal"` (или значением этого атрибута, совпадающим со вторым аргументом метода данного поведения), а также с атрибутом `href`, указывающим на URL для загрузки режимного

окна. Если же режимное окно предполагается воспроизвести в элементе разметки `iframe`, необходимо добавить также атрибут `rel` со сведениями об обработчике событий (`handler`) и размерах окна (`size`).

- Создать представление и компоновку для содержимого режимного окна. Они создаются как и обычные программные средства представления и компоновки.

Рассмотрим подробнее оба упомянутых выше примера, чтобы стало понятнее, каким образом действует поведение `modal`.

Экран конфигурации пользователя в режимном окне

Этот экран открывается в режимном окне при выборе кнопки Options на панели инструментов экрана User Manager. На рис. 12.5 приведен пример такого экрана в режимном окне.

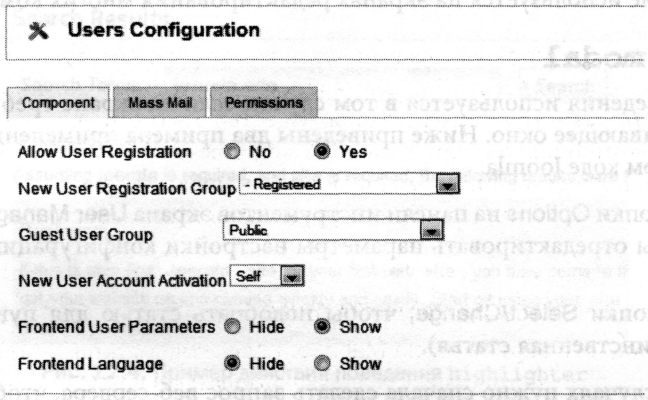


Рис. 12.5. Пример экрана конфигурации пользователя в режимном окне

Напомним, что функции кнопки Options реализуются в коде метода `addToolbar()` из класса `UsersViewUsers`, объявляемого в файле `administrator/components/com_users/views/users/view.html.php`. В этом методе можно обнаружить следующую строку кода:

```
JToolBarHelper::preferences('com_users');
```

В свою очередь, в методе `preferences()` из класса `JToolBarHelper` имеется приведенная ниже строка кода.

```
$bar->appendButton('Popup', 'options', $alt,
↳'index.php?option=com_config&amp;amp;view=component&amp;amp;component='.
↳$component.'&amp;amp;path='.$path.'&amp;amp;tmpl=component', $width, $height,
↳$top, $left, $onClose);
```

Это приводит к вызову метода `fetchButton()` из класса `JButtonPopup`, объявляемого в файле `libraries/joomla/html/toolbar/button/popup.php`. В качестве четвертого аргумента этому методу передается URL, указывающий на исходную компоновку представления компонента `com_config`. В методе `fetchButton()` можно обнаружить следующую строку кода для ввода поведения `modal`.

```
JHtml::_('behavior.modal');
```

При вызове метода поведения в этой строке кода второй аргумент опускается, и поэтому по умолчанию принимается значение `a.modal`. Далее в методе `fetchButton()` можно обнаружить приведенный ниже фрагмент кода.

```
$html = "<a class=\"modal\" href=\"$doTask\"
↳rel=\"{handler: 'iframe', size: {x: $width, y: $height},
↳onClose: function() {".$onClose."}}\">\n";
    $html .= "<span class=\"$class\">\n";
```

В этом фрагменте кода создается элемент привязки `a` с атрибутом `class="modal"` и атрибутом `href`, указывающим на URL для загрузки режимного окна и создания дополнительной функции `onClose`. Это дает возможность вызвать соответствующую функцию JavaScript при закрытии режимного окна. В данном случае функция пуста, а следовательно, при закрытии режимного окна никакой дополнительной обработки не происходит.

В конечном итоге компоновка из файла `administrator/components/com_config/views/component/tmpl/default.php` воспроизводится в режимном окне, открывающемся в элементе разметки `iframe`. Преимущество такого подхода заключается в том, что код данной компоновки очень похож на код обычной компоновки, несмотря на то, что он выполняется в режимном окне. Файл компоновки открывается при выборе кнопки Options, а после закрытия режимного окна управление возвращается родительскому окну.

Экран выбора статьи в режимном окне

Этот экран подобен экрану Article Manager (Диспетчер статей), за исключением того, что он действует в режимном окне. Он дает пользователю возможность найти и выбрать одну статью. А применяется он, в частности, при создании пункта меню Single Article, как показано на рис. 12.6.

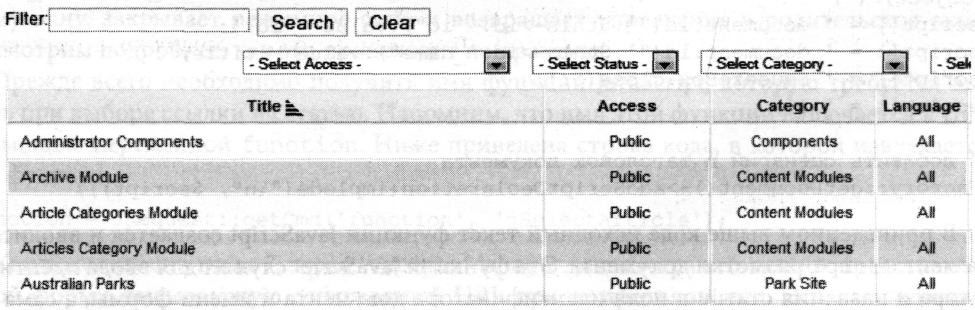


Рис. 12.6. Пример экрана выбора статьи в режимном окне

На веб-сайте могут находиться тысячи статей, которые неудобно перечислять в одном раскрывающемся списке. Поэтому в данном случае пользователю предоставляется возможность выбрать статью во всплывающем режимном окне с теми же функциями разбиения на страницы, поиска и фильтрации, что и на экране Article Manager. Таким образом,

пользователь может найти нужную ему статью среди множества других. Следует, однако, иметь в виду, что данный пример немного отличается от предыдущего, поскольку в нем требуется передавать информацию о выбранной статье обратно в родительское окно. Рассмотрим подробнее, как это делается.

Экран Single Item Menu Item (Пункт меню для выбора единственного элемента содержимого) представляет собой объект типа `JForm`, использующий XML-файл `components/com_content/views/article/tmpl/default.xml` для разметки формы. Ниже приведен элемент разметки, воспроизводящий режимное окно Select/Change Article (Выбор/Изменение статьи).

```
<fields name="request">
  <fieldset name="request"
    addfieldpath="/administrator/components/com_content/models/fields">
      <field name="id" type="modal_article"
          label="COM_CONTENT_FIELD_SELECT_ARTICLE_LABEL"
          required="true"
          description="COM_CONTENT_FIELD_SELECT_ARTICLE_DESC"
      />
  </fieldset>
</fields>
```

Атрибут `type` приведенного выше элемента разметки служит для загрузки поля из метода `getInput()` класса `JFormFieldModal_Article`, объявляемого в файле `administrator/components/com_content/models/fields/modal/article.php`. В этом методе можно обнаружить код, предназначенный для создания в родительском окне функции JavaScript, имя которой состоит из названия `jSelectArticle_` и атрибута идентификатора формы. В данном примере эта функция называется `jSelectArticle_jform_request_id`. Ниже приведен соответствующий исходный код.

```
// построить сценарий
$script = array();
$script[] = ' function jSelectArticle_'.$this->id.'(id, title, catid,
↳object) {';
$script[] = ' document.id("'.$this->id.'_id").value = id;';
$script[] = ' document.id("'.$this->id.'_name").value = title;';
$script[] = ' SqueezeBox.close();';
$script[] = ' }';

// добавить сценарий в заголовок документа
JFactory::getDocument()->addScriptDeclaration(implode("\n", $script));
```

В приведенном выше коде исходный текст функции JavaScript создается и вводится в элемент `script` разметки документа. Эта функция JavaScript служит для ввода идентификатора и названия статьи в поля идентификатора документа и имени формы, а сама она вводится в родительском окне документа. Как будет показано далее, эта функция вызывается, когда пользователь щелкает кнопкой мыши на названии статьи в режимном окне.

Далее в исходном коде метода `getInput()` можно обнаружить приведенную ниже строку, в которой формируется ссылка на URL в элементе разметки `href`.

```
$link = 'index.php?option=com_content&amp;view=articles&amp;
↳layout=modal&amp;tmpl=component&amp;function=jSelectArticle_'.$this->id;
```

Этот URL указывает на компоновку режимного окна в представлении статей из компонента com_content. После этого следует код создания элемента привязки a.

```
$html[] = ' <a class="modal" title="'.
↳JText::_('COM_CONTENT_CHANGE_ARTICLE').'" href="'.$link.
↳'&amp;'.JSession::getFormToken().'=1"
↳'" rel="{handler: \'iframe\', size: {x: 800, y: 450}}">'.
↳JText::_('COM_CONTENT_CHANGE_ARTICLE_BUTTON').'</a>';
```

Как и следовало ожидать, элемент привязки a создается с атрибутом class="modal". А в его атрибуте rel указываются сведения, необходимые для открытия режимного окна во встраиваемом фрейме, определяемом элементом разметки iframe. Обратите внимание на то, что в URL вводится переменная function с именем функции JavaScript из родительского окна (jSelectArticle_jform_request_id). В документ, представляемый в режимном окне, эта функция JavaScript не входит. Так для чего же она нужна? Дело в том, что имя этой функции должно быть известно при создании элемента привязки a для каждой статьи, перечисляемой в режимном окне. Обратите также внимание на то, что в URL вводится маркер формы. Этим гарантируется, что данная форма будет вызываться в сеансе работы в среде Joomla.

Итак, код для открытия режимного окна в родительском окне и обработки возвращаемых из него значений получен. Теперь требуется лишь код для компоновки самого режимного окна. В данном случае компоновка режимного окна выбирается из файла administrator/components/com_content/views/articles/tmpl/modal.php. Код этой компоновки очень похож на код компоновки экрана Article Manager из файла administrator/components/com_content/views/articles/tmpl/default.php, а также экрана Weblinks Manager, рассматривавшегося в главе 7. Главное отличие данной компоновки заключается в том, что происходит, когда активизируется ссылка на статью. Когда пользователь выбирает ссылку с названием статьи на экране Article Manager, эта статья открывается для редактирования. А если ссылка выбирается в режимном окне, то из родительского окна вызывается функция JavaScript под названием jSelectArticle_jform_request_id. Эта функция передает идентификатор и название статьи родительскому окну, закрывает режимное окно и возвращает управление в родительское окно. Рассмотрим подробнее, как это происходит на практике.

Прежде всего необходимо получить имя функции JavaScript, которую требуется вызвать при выборе ссылки на статью. Напомним, что имя этой функции включается в URL с помощью переменной function. Ниже приведена строка кода, в которой извлекается имя функции.

```
$function = JRequest::getCmd('function', 'jSelectArticle');
```

Как пояснялось в главе 7, экран Weblinks Manager диспетчера веб-ссылок представляет собой форму, вызывающую собственный URL при ее предъявлении. Это дает возможность воспользоваться средствами поиска и фильтрации на данном экране. Аналогичный способ применяется и на экране выбора статьи в режимном окне, как показано в приведенном ниже коде.

```
<form action="<?php echo
↳JRoute::_('index.php?option=com_content&view=articles&layout=modal
↳&tmpl=component&function='.$function.'&'.JSession::getFormToken().'=1');
↳?>" method="post" name="adminForm" id="adminForm">
```

В этом коде имя функции JavaScript, включаемое в URL, выделено полужирным. А кроме того, в URL вводится маркер формы.

Компоновка режимного окна отличается от компоновки экрана Article Manager, главным образом, ссылкой на каждую статью. Эта ссылка формируется в цикле `foreach`, как показано в приведенном ниже коде.

```
<a class="pointer" onclick="if (window.parent)
↳window.parent.<?php echo $this->escape($function);?>
↳('<?php echo $item->id; ?>',
↳'<?php echo $this->escape(addslashes($item- >title)); ?>',
↳'<?php echo $this->escape($item->catid); ?>');">
<?php echo $this->escape($item->title); ?></a>
```

В этом коде имя функции JavaScript указывается в атрибуте `onclick` элемента привязки `a` для обработки соответствующего события. В данной функции проверяется существование родительского окна, а по существу, нахождение в режимном окне. Если родительское окно существует, то вызывается функция `window.parent._jSelectArticle_jform_request_id`, которая, в свою, очередь вызывает функцию `jSelectArticle_jform_request_id` в родительском окне. Этой функции в качестве аргументов передаются идентификатор и название текущей статьи, а также идентификатор ее категории. Но на самом деле в данной функции используются лишь два первых аргумента, тогда как аргумент с идентификатором категории не принимается во внимание. Следует, однако, иметь в виду, что родительское окно загружается в оперативную память на все время работы в режимном окне. Следовательно, после того как пользователь выберет нужную статью (или закроет режимное окно), обработка в родительском окне будет продолжена в том месте, где режимное окно было открыто.

В конечном итоге режимное окно всплывает во встраиваемом фрейме, что дает пользователю возможность выбрать статью. В состав режимного окна включаются элементы управления фильтрацией и разбиением на страницы, с помощью которых пользователь может найти и выбрать нужную ему статью, даже если на веб-сайте находится много статей. Код компоновки экрана в режимном окне мало чем отличается от кода компоновки экрана соответствующего диспетчера, что упрощает его понимание и сопровождение. Аналогичные экраны в режимных окнах используются для выбора модулей, пользователей и контактной информации.

Виды поведения `checkall` и `multiselect`

Эти виды поведения дают пользователю возможность установить или сбросить сразу все или часть флажков в форме. Так, поведение `checkall` позволяет ввести в форму элемент управления для одновременной установки или сброса в ней всех флажков, тогда как поведение `multiselect` — лишь определенную часть расположенных в ряд флажков.

Одновременная установка всех флажков

На рис. 12.7 приведен пример элемента управления одновременной установкой или сбросом всех флажков на экране Weblinks Manager. Этот элемент управления обозначен на рисунке стрелкой.

Для реализации такого поведения выполните следующие действия.

1. Загрузите файл `core.js`, содержащий, например, вызов метода поведения `JHtml::_('behavior.framework')` или какого-нибудь другого поведения, предполагающего загрузку интегрированной среды MooTools.
2. Введите в разметку формы элемент управления вводом данных типа флажка, присвоив атрибуту `onclick` значение `"Joomla.checkAll(this)"`.
3. Дополните значение атрибута `id` буквами `"cb"` в разметке всех одновременно устанавливаемых флажков. Например, в экранных формах базовых диспетчеров Joomla каждый ряд флажков обозначается стандартной меткой `"cbx"`, где `x` — номер ряда от **0** до **n**.

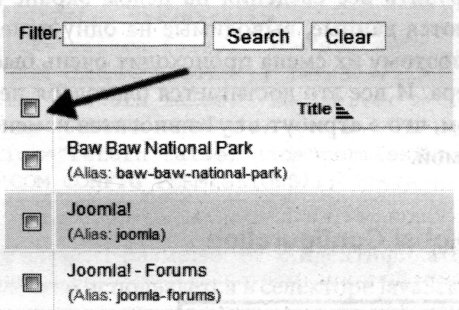


Рис. 12.7. Пример реализации поведения для одновременной установки или сброса всех флажков в форме

Одновременная установка части флажков

Поведение `multiselect` позволяет устанавливать или сбрасывать сразу лишь определенную часть флажков одновременным нажатием клавиши <Shift> и щелчком левой кнопкой мыши. Так, если все флажки сброшены и требуется установить только флажки `2`–**8**, для этого достаточно нажать клавишу <Shift> и щелкнуть сначала на флажке `2`, а затем на флажке **8**. Аналогичным способом можно сбросить определенную часть расположенных в ряд флажков. Для того чтобы активизировать поведение `multiselect`, достаточно ввести вызов его метода `JHtml::_('behavior.multiselect')` в исходный код соответствующего сценария.

Поведение `noframes`

Этот вид поведения предотвращает загрузку страницы во встраиваемый фрейм, чтобы обезопасить веб-сайт от атак в целях фишинга. Например, ложная ссылка может направить пользователя на веб-сайт, где экран регистрации действует во встраиваемом фрейме. Для пользователя этот экран будет выглядеть точно так же, как и экран настоящего веб-сайта, а злоумышленники смогут собрать секретные учетные данные регистрации данного пользователя. Назначение поведения `noframes` в том и состоит, чтобы предотвратить подобного рода подлог. Для реализации этого поведения достаточно ввести вызов его метода `JHtml::_('behavior.noframes')` в исходный код программы.

Поведение switcher

Этот вид поведения дает разработчикам возможность создавать похожие на вкладки панели в интерфейсе HTML-страницы. Такая возможность оказывается удобной, когда требуется отобразить дополнительные сведения на экране, вписывающемся в одну страницу. В качестве примера на рис. 12.8 показан экран Global Configuration, на котором применяется поведение switcher.

По своим функциям это поведение аналогично подменю. После выбора каждой ссылки отображаются сведения, связанные с соответствующей панелью формы, определяемой элементом div в разметке документа. Отличие заключается в том, что поведение switcher позволяет загрузить все сведения на целом экране в одном цикле запроса. Одновременно отображаются данные, выводимые на одну панель, но в документе присутствуют все панели, и поэтому их смена происходит очень быстро, не требуя загрузки новой панели с веб-сервера. И все это достигается благодаря ненавязчивости JavaScript, которая проявляется в том, что в атрибут style вносятся изменения с целью скрыть все панели, кроме отображаемой.

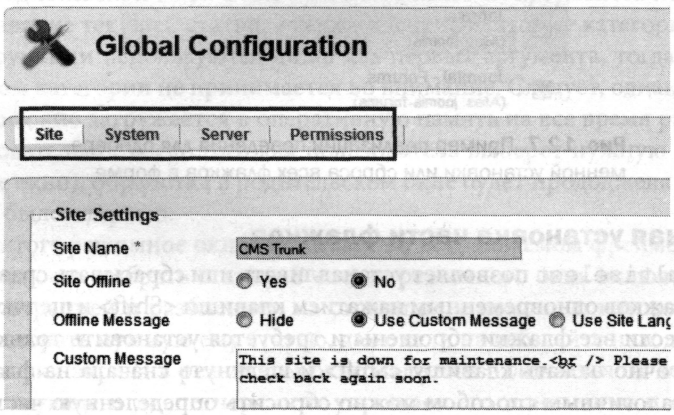


Рис. 12.8. Пример экрана, реализующего поведение switcher

Рассмотрим подробнее действие данного вида поведения на примере экрана Global Configuration. Если заглянуть в папку administrator/components/com_config/views/application/tmpl, то в ней можно обнаружить 17 различных файлов компоновки. Первым из них загружается файл компоновки default.php. Ниже приведена первая часть этого файла.

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

// загрузить поведение всплывающих подсказок
JHtml::_('behavior.formvalidation');
JHtml::_('behavior.switcher');
JHtml::_('behavior.tooltip');

// загрузить шаблон подменю, используя идентификатор элемента 'submenu',
// требующийся для метода поведения behavior.switcher
```

```
$this->document->setBuffer($this->loadTemplate('navigation'), 'modules',
↳'submenu');
```

В приведенный выше код сначала вводится вызов метода поведения `behavior.switcher` для загрузки требующегося сценария JavaScript. Затем в выделенной полужирным строке кода формируются ссылки на подменю. В этой строке кода выполняется файл компоновки `default-navigation.php`, а результат выводится на месте расположения модуля, обозначаемом как `submenu`.

В файле `default-navigation.php` содержится следующий код:

```
<ul id="submenu" class="configuration">
  <li><a href="#" onclick="return false;" id="site" class="active">
↳<?php echo JText::_('JSITE'); ?></a></li>
  <li><a href="#" onclick="return false;" id="system">
↳<?php echo JText::_('COM_CONFIG_SYSTEM'); ?></a></li>
  <li><a href="#" onclick="return false;" id="server">
↳<?php echo JText::_('COM_CONFIG_SERVER'); ?></a></li>
  <li><a href="#" onclick="return false;" id="permissions">
↳<?php echo JText::_('COM_CONFIG_PERMISSIONS'); ?></a></li>
</ul>
```

В этом коде создается элемент разметки `ul` с идентификатором подменю, что очень важно. Ведь это обстоятельство используется в селекторе JavaScript для указания на то, что данный список ссылок будет служить для смены панелей. А в каждом элементе разметки `li` определяется ссылка на одну из панелей, выводимых на экран с поведением `switcher`. Имя панели указывается в атрибуте `id`. В определении первой ссылки атрибуту `class` присваивается значение `active`, как выделено полужирным в приведенном выше коде. Этим пользователю дополнительно указывается на то, что данная панель активна.

А теперь вернемся к файлу компоновки `default.php`. В следующей его части находится код для ввода сценария JavaScript, активизирующего рассматривавшееся ранее поведение для проверки достоверности данных в форме. Далее следует приведенный ниже код.

```
<form action="<?php echo JRoute::_('index.php?option=com_config');?>"
↳id="application-form" method="post" name="adminForm"
↳class="form-validate">
    <?php if ($this->ftp) : ?>
      <?php echo $this->loadTemplate('ftplogin'); ?>
    <?php endif; ?>
    <div id="config-document">
      <div id="page-site" class="tab">
        <div class="noshow">
          <div class="width-60 fltlft">
            <?php echo $this->loadTemplate('site'); ?>
            <?php echo $this->loadTemplate('metadata'); ?>
          </div>
          <div class="width-40 fltrt">
            <?php echo $this->loadTemplate('seo'); ?>
            <?php echo $this->loadTemplate('cookie'); ?>
          </div>
        </div>
    </div>
```

В первой строке этого кода создается обычная форма. Затем загружается страница регистрации на FTP-сервере, если такая возможность имеется. В двух последующих строках кода выполняются действия, очень важные для поведения `switcher`, поэтому они и выделены полужирным. В первой из них создается элемент разметки `div` с атрибутом `id="config-document"`. Этот элемент разметки служит контейнером для всех панелей, которые будут действовать под управлением поведения `switcher`. Во второй строке кода создается первая панель. Для этой цели служит элемент разметки `div` с атрибутом `id`, имеющим значение `page-xxx`, где `xxx` совпадает с идентификатором элемента привязки `a` в разметке подменю. Данной панели необходимо также присвоить класс вкладки, что для чего служит атрибут `class="tab"`.

Далее следуют четыре выделенные полужирным строки кода В каждой из них компоновка загружается на нужное место страницы. Например, первой загружается компоновка из файла `default-site.php`, а второй — компоновка из файла `default-metadata.php`.

После этого следует приведенная ниже строка кода.

```
<div id="page-system" class="tab">
```

В этой строке кода создается вторая панель, соответствующая ссылке на систему в подменю. Еще далее создается панель для страницы сервера (`page-server`), а затем панель для страницы полномочий (`page-permissions`). Для каждой из них загружается один или более сценарий компоновки. Когда страница загружается в первый раз, функция JavaScript проверяет, какая именно панель должна отображаться. Для запоминания пользователя, работавшего с панелью в последний раз, используется cookie-файл. В отсутствие cookie-файла по умолчанию отображается первая панель.

Функция JavaScript действует следующим образом. В ней создается массив элементов разметки `div` с атрибутом `class="tab"`. Эти элементы разметки служат контейнерами для панели каждой вкладки. Так, для отображаемой панели в данной функции динамически создается атрибут `style = "display: block;"`, а для всех остальных панелей — атрибут `style = "display: none;"`. Когда пользователь щелкает кнопкой мыши на ссылке подменю, данный процесс повторяется. Элемент разметки `div`, соответствующий выбранной пользователем ссылке, динамически обновляется (его атрибуту `style` присваивается значение `"display: block;"`), а следовательно, определяемая им панель отображается, тогда как остальные панели скрываются (атрибутам `style` их элемента разметки `div` присваивается значение `"display: none;"`).

Это относительно простой и удобный способ реализации подобного рода поведения. Ведь файл компоновки каждой панели очень прост, что позволяет легко изменить расположение вкладок на странице или создать новые вкладки. Подобным способом организуется экран, вызываемый по команде Site⇨System Information в базовом коде Joomla.

Ниже приведены основные требования к применению поведения `switcher`.

- Ввести поведение `switcher` в компоновку.
- Создать компоновку, аналогичную рассмотренной ранее компоновке из файла `default-navigation.php`. Она должна содержать элемент разметки `ul` с атрибутами `id="submenu"` и `class="configuration"`. В этот элемент следует также ввести элементы разметки `li` с элементами привязки `a`, указав имя панели (например, `"site"`, `"system"` и т.д.) в атрибуте `id` каждого элемента привязки.

- Загрузить эту компоновку для перемещения по панелям из исходной компоновки (например, на месте модуля подменю).
- Создать еще одну или несколько компоновок для каждой панели.
- Создать в исходной компоновке элемент разметки `div` с атрибутом `id="config-document"`.
- Создать в этом элементе по одному элементу разметки `div` на каждую панель. Каждый из этих элементов разметки должен содержать атрибут `id` со значением `"page-xxx"`, а также атрибут `class` со значением `"tab"`.
- Организовать вызов метода `loadTemplate()` в каждом из упомянутых выше элементов разметки `div`, чтобы загрузить файлы компоновки отдельных частей страницы.

Поведение `tooltip`

Этот вид поведения позволяет создавать подсказки, всплывающие при наведении пользователем курсора на элемент формы. Такое поведение обычно используется в Joomla для предоставления вспомогательного текста с кратким описанием элемента, как показано на рис. 12.9.

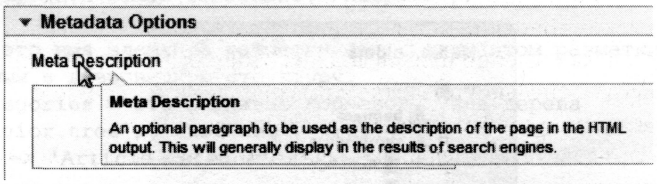


Рис. 12.9. Пример всплывающей подсказки

Для реализации всплывающей подсказки достаточно вызвать поведение `tooltip` и создать элемент разметки с атрибутом `class="hasTip"`, а также атрибутом `title` с текстом, появляющимся во всплывающей подсказке. К дополнение к этому можно также создать заголовок и основной текст для всплывающей подсказки, используя атрибут `title` в следующем формате:

```
<заголовок>::<основной текст>
```

Например, атрибут `title` с текстом, приведенным во всплывающей подсказке на рис. 12.9, определяется следующим образом:

```
title="Meta Description::An optional paragraph to be used as the
↳description of the page in the HTML output. This will generally display
↳in the results of search engines."
```

Всплывающие подсказки создаются автоматически средствами класса `JForm`. В этом случае атрибут `label` служит для заголовка подсказки, тогда как атрибут `description` — для текста описания во всплывающей подсказке. Напомним, что если значения этих атрибутов служат в качестве ключей в файле языковой поддержки, то они будут заменены переводом соответствующего текста на избранный язык во всплывающей подсказке.

Создать всплывающую подсказку можно также, вызвав метод поведения `JHtml::tooltip()`. Например, в следующей строке кода:

```
echo JHtml::_('tooltip', 'My tooltip text.', 'My Tooltip Title', '',
↳'Hover on this text to see the tooltip.');
```

создается приведенная ниже разметка всплывающей подсказки в формате HTML.

```
<span class="hasTip" title="My Tooltip Title::My tooltip text.">Hover on
↳this text to see the tooltip.</span>
```

Поведение tree

Этот вид поведения используется на экране Media Manager для отображения папок с мультимедийными файлами в виде древовидной структуры. Щелкнув на пиктограмме со знаком + или -, пользователь может развернуть или свернуть древовидную структуру папок. На рис. 12.10 приведен пример полностью развернутой древовидной структуры папок на экране Media Manager.

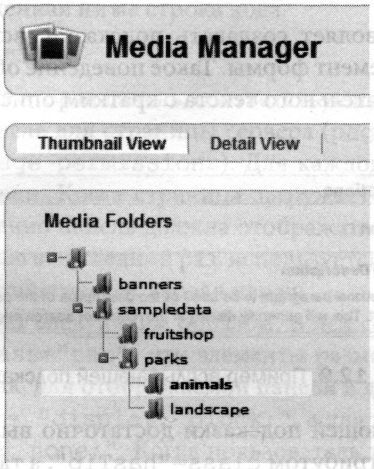


Рис. 12.10. Пример древовидной структуры, реализуемой с помощью поведения `tree`

Поведение `tree` действует следующим образом. В компоновке создается элемент `ul` разметки неупорядоченного списка с элементом привязки `a` в каждом элементе `li` разметки списка. Эти элементы разметки допускают вложение, чтобы в элементе `li` можно было бы определить другой элемент `ul` и т.д. Ниже приведен пример разметки древовидной структуры.

```
<ul id="testtree">
    <li><a href="test.html">Item One</a></li>
    <li><a href="test.html">Item Two</a>
        <ul>
            <li><a href="test.html">Item Two Point One</a></li>
            <li><a href="test.html">Item Two Point Two</a></li>
        </ul>
    </li>
    <li><a href="test.html">Item Three</a>
    </li>
</ul>
```

Функция JavaScript считывает внешний элемент разметки ul, исходя из значения его атрибута id, создает соответствующий объект древовидной структуры и скрывает весь элемент разметки ul. В итоге появляется древовидная структура, но не список. Значения привязки превращаются в ссылки на URL, используемые при выборе элемента древовидной структуры. Если элемент разметки li вложен в элемент разметки ul, то в результате щелчка кнопкой мыши на пиктограмме со знаком + или - вложенный список разворачивается или сворачивается.

Древовидные структуры могут оказаться полезными для работы с вложенными иерархиями, в том числе файлов или категорий. Покажем в качестве примера, как воспользоваться поведением tree для отображения категорий статей в виде древовидной структуры. С этой целью создадим замещение шаблона для элемента меню Articles⇨List all Categories (Статьи⇨Перечислить все категории) и внесем изменения в его компоновку, чтобы показать простую древовидную структуру. Для такого замещения нужно сначала создать папку templates/beez_20/html/com_content/categories, а затем файл компоновки default.php со следующим кодом:

```
<?php
// прямой доступ отсутствует
defined('_JEXEC') or die;
JHtml::addIncludePath(JPATH_COMPONENT.'/helpers');

// testtree - это имя элемента разметки div с элементом разметки ul,
// преобразуемым в древовидную структуру.
// Article Categories - это заглавие корневого узла дерева
JHtml::_('behavior.tree', 'testtree', array(),
↳array('text' => 'Article Categories'));
?>

<h1>MooTree Example Template Override</h1>
<div id="testtree_tree"></div>

<?php echo $this->loadTemplate('items'); ?>
```

В данном файле компоновки вызывается метод поведения behavior.tree с тремя аргументами. В качестве первого его аргумента указывается "testtree" — идентификатор элемента разметки ul, преобразуемого в дерево; в качестве второго аргумента — массив array() с дополнительными параметрами, которые в данном случае не используются; в качестве третьего аргумента — текст заголовка в самом верхнем узле дерева (он выбирается из ассоциативного массива по текстовому ключу). По умолчанию используется заголовок "Root" (Корневой узел), но в данном случае — заголовок "Article Categories" (Категории статей).

В рассматриваемом здесь файле компоновки создаются элемент разметки заголовка и пустой элемент разметки div с атрибутом id="testtree_tree". Такой элемент разметки div потребуется в дальнейшем для создания древовидной структуры. Значение атрибута id этого элемента разметки состоит из идентификатора элемента разметки ul и суффикса _tree (в данном случае — testtree_tree).

И в последней строке кода из данного файла компоновки вызывается другой файл компоновки — `default_items.php`. Этот файл компоновки создается в том же самом каталоге. Ниже приведено его содержимое.

```
<?php
// прямой доступ отсутствует
defined('_JEXEC') or die;
?>
<ul id="testtree">
<?php foreach($this->items[$this->parent->id] as $id => $item) : ?>
    <li><a href="<?php echo
↳JRoute::_(ContentHelperRoute::getCategoryRoute($item->id));?>">
        <?php echo $this->escape($item->title); ?> </a>
    <?php if ($children = $item->getChildren(false)) : ?>
        <ul>
          <?php foreach ($children as $child) : ?>
            <li><a href="<?php echo
↳JRoute::_(ContentHelperRoute::getCategoryRoute($child->id));?>">
                <?php echo $this->escape($child->title); ?> </a>
            </li>
        <?php endforeach; ?>
        </ul>
    <?php endif; ?>
    </li>
<?php endforeach; ?>
</ul>
```

Сначала в этом файле компоновки создается элемент разметки `ul` с атрибутом `id="testtree"` (выделено выше полужирным) для построения древовидной структуры. С этой целью организуются два цикла `foreach`. Во внешнем цикле создается элемент `li` разметки каждой категории по списку, а для привязки к URL используется маршрут категории. Затем проверяется наличие подчиненных категорий для каждой категории. Если таковые имеются, то для разметки подчиненных категорий создается внутренний элемент `ul`. Следует заметить, что в данном примере древовидная структура развертывается только на один уровень вниз, хотя аналогичным образом ее можно развернуть на любое число уровней.

Для того чтобы посмотреть построенную таким образом древовидную структуру в действии, достаточно воспользоваться выборочными данными для версии Joomla 2.5 и перейти по команде Home⇨Using Joomla!⇨Using Extensions к папке `book_170/index.php/using-joomla/extensions`. На рис. 12.11 приведен пример компоновки замещения шаблона с поведением `tree`.

Поведение `uploader`

Этот вид поведения используется на экране Media Manager, если кнопка-переключатель Enable Flash Uploader (Активизировать Flash-загрузчик) установлена в положение Yes. Поведение `uploader` позволяет отображать сведения о файлах, выгружаемых на веб-сервер, а также ход выполнения данного процесса. На рис. 12.12 приведен пример соответствующего экрана.

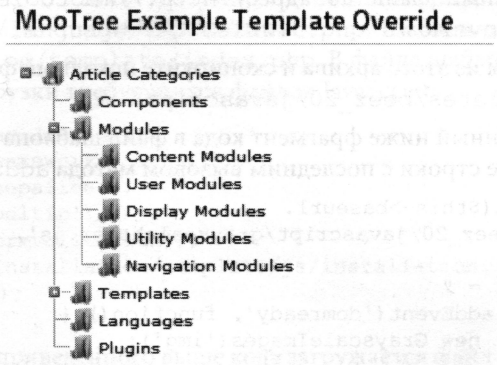


Рис. 12.11. Пример компоновки замещения шаблона с поведением `tree`

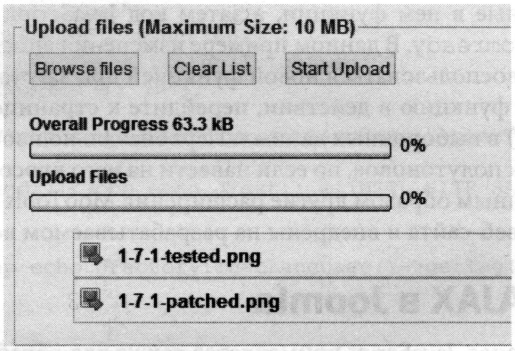


Рис. 12.12. Пример экрана с состоянием выгрузки файлов в компоненте Media Manager

Данное поведение тесно связано с компонентом Media Manager и не предназначено для общего пользования. Но в состав библиотеки MooTools входит целый ряд элементов управления полосами состояния и связанных с ними виджетов для показа пользователям состояния процесса, выполняемого в фоновом режиме.

Применение расширений MooTools

Помимо базовых функциональных средств MooTools, поддерживается веб-сайт по адресу `http://mootools.net/forge`, где перечислены многие расширения базовой среды MooTools. Эти расширения могут быть загружены и использованы на веб-сайте Joomla. Продемонстрируем их возможности на примере простого в употреблении расширения Grayscale Image Hover, написанного Рувеном Уэсслингом (Rouven Wessling). Это расширение позволяет изменять полутоновое изображение на цветное при наведении на него курсора мыши. Ниже приведена последовательность действия для внедрения этого расширения в исходный код разрабатываемого приложения.

1. Загрузите архивный файл по адресу `http://mootools.net/forge/p/grayscale_image_hover`.
2. Извлеките файлы из этого архива и скопируйте исходный файл `grayscalehover.js` в папку `templates/beez_20/javascript`.
3. Введите приведенный ниже фрагмент кода в файл шаблона `templates/beez_20/index.php` после строки с последним вызовом метода `addScript()`.

```
$doc->addScript($this->baseurl.
↳'/templates/beez_20/javascript/grayscalehover.js', 'text/javascript',
↳true);
$grayscaleHover = "
        window.addEvent('domready', function() {
                new GrayscaleImages('img');
        });
";
$doc->addScriptDeclaration($grayscaleHover);
```

В этом фрагменте кода сначала вводится новый исходный файл JavaScript, чтобы использовать определенные в нем функции, а затем код JavaScript, чтобы связать новую функцию с событием `domready`. В данном примере изменения вносятся в каждый элемент разметки `img` с целью воспользоваться новой функцией при загрузке страницы. Для того чтобы посмотреть эту функцию в действии, перейдите к странице `Australian Parks` (Австралийские парки) в выборочных данных. Первоначально изображение показывается на этой странице как полутоновое, но если навести на него курсор, оно постепенно станет цветным. Аналогичным образом другие расширения MooTools могут быть загружены из упомянутого выше веб-сайта и внедрены на разрабатываемом веб-сайте Joomla.

Применение AJAX в Joomla

Как упоминалось ранее, JavaScript применяется также для обработки асинхронных запросов к серверу. Преимущество таких запросов заключается в том, что они могут быть выполнены в фоновом режиме и не требуют обновления всей страницы. Сценарий JavaScript выполняется в браузере непрерывно и поэтому способен управлять всем процессом.

Подобная технология называется AJAX и применяется, например, при установке Joomla. Подробное описание процесса создания AJAX-ориентированных приложений выходит за рамки этой книги, и поэтому здесь нет места для детального анализа кода JavaScript. В связи с этим применение AJAX в Joomla будет рассмотрено лишь в самых общих чертах, но с акцентом на взаимодействие кода JavaScript и PHP.

Программы установки Joomla представляют собой отдельное приложение, находящееся в специально выделенной для него папке. Такие программы обычно выполняются один раз для установки экземпляра Joomla, а затем вся установочная папка удаляется из соображений безопасности.

Точкой входа в установку нового экземпляра Joomla служит файл `installation/index.php`, который, в свою очередь, запускает на выполнение файл `installation/controller.php`. В этом файле сначала загружаются требующиеся классы платформы и переменные окружения, указывающие пути к файлам, а затем устанавливается исходное представление для выбора языка (первый экран, появляющийся при установке Joomla). Для этой цели выполняется представление типа `JInstallationViewLanguage`

из файла installation/views/language/view.html.php, а также компоновка installation/views/language/tmp/default.php. Шаблон для установки находится в файле installation/template/index.php. В файле шаблона index.php имеется следующий код для загрузки требующихся файлов JavaScript:

```
// загрузить поведения JavaScript
JHtml::_('behavior.framework', true);
JHtml::_('behavior.keepalive');
JHtml::_('behavior.tooltip');
JHtml::_('behavior.formvalidation');
JHtml::_('script', 'installation/template/js/installation.js', true,
↳false, false, false);
?>
```

В последней строке приведенного выше кода загружается файл installation.js, используемый для управления всем процессом установки. Далее в файле шаблона index.php можно обнаружить приведенный ниже фрагмент кода JavaScript, размещаемый в элементе разметки script.

```
<script type="text/javascript">
  window.addEvent('domready', function() {
    window.Install = new Installation('rightpad',
↳'<?php echo JURI::current(); ?>');

    Locale.define('<?php echo JFactory::getLanguage()->getTag(); ?>',
↳'installation', {
      sampleDataLoaded: '<?php echo JText::_('INSTL_SITE_SAMPLE_LOADED',
↳true); ?>'
    });
    Locale.use('<?php echo JFactory::getLanguage()->getTag(); ?>');
  });
</script>
```

Сначала в этом фрагменте кода функция JavaScript связывается с событием domready, которое инициируется, как только HTML-документ будет полностью загружен. Затем в этом коде создается новый объект Install типа Installation по сценарию JavaScript из файла installation/template/js/installation.js. Этот сценарий берет на себя дальнейшее управление всем процессом установки. В правом верхнем углу экрана представления для выбора языка имеется кнопка **Next** (Далее), как показано на рис. 12.13. В разметке этой кнопки атрибуту onclick присвоено имя метода Install.submitform(). Этот метод выполняется для объекта Install, создаваемого по сценарию JavaScript при наступлении события domready в связи с загрузкой страницы.

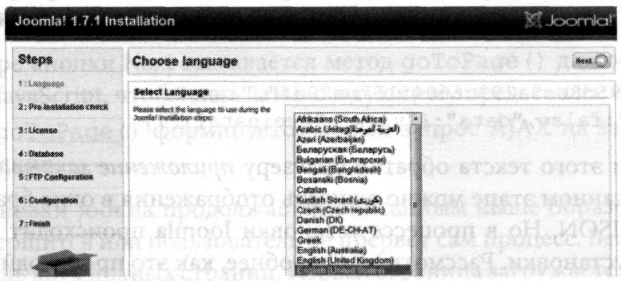


Рис. 12.13. Экран выбора языка при установке Joomla

На данном этапе управление процессом установки переходит к объекту `Install` по сценарию JavaScript из упомянутого выше файла. Вместо загрузки страниц по отдельным запросам, отправляемым из браузера на сервер, объект `Install` посылает асинхронные запросы на сервер, чтобы обновить имеющиеся на странице данные. Метод `submitform()` из объекта `Install` считывает задачи из элемента разметки `form` с атрибутом `name="task"`, используя эти сведения для ввода задачи в запрос. В первый запрос, посылаемый объектом `Install`, вводится задача `task=setup.setlanguage&format=json`. Этот запрос посылается после того, как пользователь щелкнет на кнопке Next в правом верхнем углу экрана выбора языка (см. рис. 12.13), но посылается он на сервер асинхронно. Это означает, что сценарий JavaScript продолжает работать в то время, как на сервере выполняется задача по данному запросу.

Этот запрос обрабатывается сервером, как и обычный запрос от браузера, после чего процесс установки возвращается к выполнению обычных программ Joomla на PHP. В данном случае выполняется метод `setlanguage()` из класса `JInstallationControllerSetup`, объявляемого в файле `installation/controllers/setup.json.php`. Следует заметить, что указание формата `format=json` в URL по данному запросу приводит к загрузке файла `setup.json.php`. В методе `setlanguage()` получаются некоторые данные из модели, проверяются ошибки, а завершается он следующим фрагментом кода:

```
// переадресовать на следующую страницу
$r->view = 'preinstall';
$this->sendResponse($r);
```

В этом фрагменте кода переменная `$r` ссылается на поле `view` объекта типа `Jobject`, в котором устанавливается имя `preinstall` следующего представления в процессе установки Joomla. Затем вызывается метод `sendResponse()`. Именно здесь и начинается самое интересное. В методе `sendResponse()` проверяются ошибки, а завершается он следующим фрагментом кода:

```
// отправить запрос в формате JSON
echo json_encode(new JInstallationJsonResponse($response));

// закрыть приложение
$app = JFactory::getApplication();
$app->close();
```

В этом фрагменте кода сначала создается новый объект типа `JInstallationJsonResponse`, а затем он передается обратно браузеру закодированным в формате JSON. В данном случае текст в формате JSON содержит сеансовый маркер, язык, состояние ошибки и данные с именем следующего отображаемого представления аналогично приведенному ниже.

```
{"token":"c8de8295d8be5a299c66045157ba29df","lang":
↳"en-US","error":false,"data":{"view":"preinstall"}}
```

После передачи этого текста обратно браузеру *приложение закрывается*! В обычном цикле запроса на данном этапе можно ожидать отображения в окне браузера некоторого текста в формате JSON. Но в процессе установки Joomla происходит переход на экран предварительной установки. Рассмотрим подробнее, как это происходит.

Для понимания рассматриваемого здесь кода PHP очень важно иметь в виду, что он выполняется под управлением класса `Installation` (а по существу, объекта `Install`) из сценария JavaScript. Запрос AJAX посылается с помощью метода `post()` из класса `Request.JSON`, входящего в библиотеку MooTools. В этом классе определяется событие `onSuccess`, инициируемое по завершении обработки данного запроса. В этот момент выполнение сценария JavaScript продолжается с кода обработки события `onSuccess`, наступающего при успешном завершении данного запроса.

Когда приложение Joomla на PHP закрывается вызовом метода `$app->close()`, управление передается обратно объекту `Install` из сценария JavaScript, а точнее, коду обработки события `onSuccess`, наступающего при успешном завершении данного запроса. Сначала в этом коде считывается символьная строка формата JSON, передаваемая обратно браузеру с помощью метода `sendResponse()`, а затем получается представление из элемента `data` в символьной строке формата JSON (в данном случае представление под именем `preinstall`) и далее вызывается метод `goToPage()`, которому в качестве аргумента передается имя данного представления. В методе `goToPage()` выдается новый запрос AJAX на загрузку представления `preinstall` в процессе установки Joomla. В представлении `preinstall` определяется кнопка **Next** для вызова `goToPage('license')` функции JavaScript. Это приводит к непосредственной загрузке следующего представления под названием `license`.

При переходе от одного экрана к другому в процессе установки Joomla соблюдается определенный порядок действий. Так, если на экране требуется выполнить задачу, отличающуюся от простого отображения, то происходит следующее.

- При выборе кнопки **Next** вызывается метод `submitform()` для объекта `Install` из сценария JavaScript. В этом методе задача считывается из элемента `task` разметки формы, который обычно скрыт.
- Метод `submitform()` посылает новый запрос AJAX на обработку данной задачи. По завершении ее обработки в программе на PHP из последней передается символьная строка в формате JSON обратно браузеру, а затем приложение Joomla на PHP закрывается.
- Инициируется событие `onSuccess`, и объект `Install` переходит к коду обработки этого события. В данном случае имя следующего представления считывается из символьной строки формата JSON, после чего данное представление загружается по следующему запросу AJAX.

Если же на экране требуется только загрузить следующее представление без дополнительной обработки, то происходит следующее.

- При выборе кнопки **Next** вызывается метод `goToPage()` для объекта `Install` из сценария JavaScript, чтобы перейти к следующему представлению.
- В методе `goToPage()` формируется новый запрос AJAX на загрузку следующего представления.

Процесс установки Joomla продолжается описанным выше образом до тех пор, пока установка не завершится или пользователь не прервет сам процесс. Весь этот процесс требует загрузки лишь двух полных страниц. Первая страница загружается с первоначальным представлением для выбора языка. А вторая загружается после того, как пользователь

удалит установочную папку и перейдет к административной или пользовательской части вновь установленного веб-сайта.

Как видите, этот процесс соответствует обычному шаблону MVC (Модель–Контроллер–Представление) в Joomla, где один цикл запроса может служить доя выполнения задачи, а другой — для отображения компоновки. Но его отличие заключается в том, что запросы делаются асинхронно, а в оперативной памяти все время находится один и тот же сценарий JavaScript, управляющий данным процессом.

Применение других интегрированных сред JavaScript

Воспользоваться другими интегрированными средами JavaScript в Joomla относительно просто. Рассмотрим эту процедуру на примере одновременного применения программных средств из распространенных интегрированных сред jQuery и MooTools на одной странице веб-сайта. Для этого достаточно выполнить следующие действия.

1. Создайте на пробном веб-сайте Joomla папку `components/com_test`. В ней будут храниться файлы, необходимые для данного примера.
2. Загрузите интегрированную среду jQuery из веб-сайта по адресу `http://docs.jquery.com/Downloading_jQuery#Download_jQuery`. В данном примере используется файл `jquery-1.6.4.js`. Скопируйте этот файл в папку `com_test`.
3. Создайте в папке `com_test` файл `noconflict.js`, введя в него следующий код:

```
/**
 * Файл noconflict.js для демонстрации загрузки jQuery
 * средствами MooTools
 */
jQuery.noConflict();
```

4. Создайте в папке `com_test` еще один файл `test.php`, введя в него следующий код:

```
<?php
// Пробный файл для демонстрации загрузки библиотек MooTools и jQuery

JHtml::_('behavior.framework');
$document = JFactory::getDocument();
$document->addScript(JURI::root(true).
'/components/com_test/jquery-1.6.4.js');
$document->addScript(JURI::root(true).
'/components/com_test/noconflict.js');

$mootoolsStyle = "
// MooTools изменяет начертание шрифта, когда наступает событие domready
    window.addEvent('domready',function() {
      $$('p.test').setStyle('font-style','italic');
  }); ";
$document->addScriptDeclaration($mootoolsStyle);

$jqueryStyle = "
```

```
// jQuery изменяет цвет фона документа, когда наступает событие domready
    jQuery(document).ready(function($){
        jQuery('p.test').css('background-color','#fafa99');
        }); ";
$document->addScriptDeclaration($jqueryStyle);
?>

<p class="test">MooTools sets this paragraph's text to italic,
and jQuery sets the background to yellow.</p>
```

В файле `test.php` сначала вызывается метод поведения `behavior.framework` для загрузки интегрированной среды MooTools, а затем метод `addScript()` из класса `JDocument` для загрузки интегрированной среды jQuery из файла `jquery-1.6.4.js`, а также файла `noconflict.js`. Второй из этих файлов обеспечивает работу jQuery в "бесконфликтном" режиме. По умолчанию для обращения к обеим интегрированным средам MooTools и jQuery используется сокращение **$**. Так, если jQuery работает в бесконфликтном режиме, то сокращение **$** может быть по-прежнему использовано в MooTools. В данном случае для обращения к функциям jQuery можно воспользоваться переменной `jQuery` или другой назначенной переменной. Очень важно, чтобы метод `addScript()` вызывался для загрузки файла `noconflict.js` сразу же после метода `addScript()`, вызываемого для загрузки файла `jquery-1.6.4.js`.

После загрузки обеих интегрированных сред и файла `noconflict.js` демонстрируется их одновременное применение на одной и той же странице веб-сайта. Для этой цели созданы две функции. В первой из них, MooTools, используется для изменения на курсив начертания шрифта, которым набран абзац текста, а во второй, jQuery, для изменения на желтый цвета фона. Обе эти функции связываются с событием `domready`, наступающим после готовности документа, а следовательно, они вызываются, как только документ будет загружен. И наконец, в исходный код рассматриваемого здесь примера включен элемент разметки абзаца `p` с атрибутом `class="test"`. Содержимое этого элемента разметки (т.е. абзац текста) изменяется в обеих упомянутых выше функциях JavaScript.

Для просмотра результатов изменений на странице загрузите пробный веб-сайт Joomla по следующему адресу: `<ваш URL>/index.php?option=com_test`. Напомним, что при этом выполняется файл `components/com_test/test.php`. В итоге должен появиться экран, приведенный на рис. 12.14. Обратите внимание на сообщения об ошибках на этом экране. Они появляются потому, что для компонента `com_test` отсутствует строка в таблице `#__extensions`.

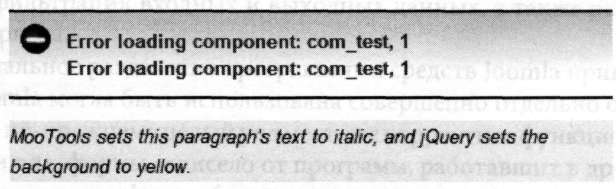


Рис. 12.14. Пример одновременного применения MooTools и jQuery

Как пояснялось ранее, для работы в Joomla лучше пользоваться MooTools, поскольку эта интегрированная среда уже включена в состав Joomla, избавляя от необходимости

загружать и сопровождать исходный код для другой интегрированной среды. Но если требуется, то наряду с MooTools можно воспользоваться и другой интегрированной средой, как показано в рассмотренном здесь примере.

Резюме

В этой главе были представлены язык программирования JavaScript и интегрированная среда MooTools для разработки программ на этом языке, а также показано, каким образом они применяются при разработке веб-сайтов на Joomla. Далее были рассмотрены отдельные средства пользовательского интерфейса, которые могут быть реализованы имеющимися методами JavaScript. И в завершение этой главы был вкратце описан порядок применения технологии AJAX в Joomla на примере программы установки.

JavaScript позволяет расширить возможности взаимодействия веб-сайтов на Joomla с пользователями и выгодно воспользоваться преимуществами технологии AJAX. Кроме того, JavaScript служит ключевым компонентом для разработки веб-служб, позволяющих веб-сайтам на Joomla предоставлять информацию другим программам, в том числе приложениям для мобильных устройств и веб-серверам. Именно по этим причинам ожидается, что в недалеком будущем приложения Joomla постепенно приобретут большее значение.

13

Применение платформы Joomla! в качестве среды для разработки приложений

После выпуска версии Joomla 1.7.0 в июле 2011 года платформа Joomla официально стала самостоятельным проектом, отдельным от системы управления контентом (CMS) веб-сайтов на Joomla. Это позволило развивать платформу Joomla независимо от CMS и упростить ее самостоятельное развертывание для разработки и усовершенствования самых разных веб-приложений.

Платформа Joomla — довольно обширная тема, вполне заслуживающая отдельной книги. Поэтому в этой главе будет сделан лишь краткий обзор данной платформы и приведен ряд простых примеров ее применения.

Общее представление о платформе Joomla

Версия Joomla 1.5 разработана таким образом, чтобы классы, выполняющие основные функции, размещались в папке `libraries/joomla`. И хотя эти классы применяются в самых разных местах CMS, они выполняют функции, общие для большинства веб-приложений, а не только для отдельного применения CMS. К числу этих функций относится аутентификация пользователей, перевод текста на другие языки, фильтрация входных и выходных данных, а также работа с файловой системой сервера.

Такое первоначальное разделение программных средств Joomla привело к тому, что платформа Joomla могла быть использована совершенно отдельно от CMS. Тем не менее оставался по-прежнему целый ряд мест, где корректное функционирование программ из папки платформы зависело от программ, работавших в других местах CMS. Но после того как платформа была выпущена в рамках отдельного проекта, эти зависимости были устранены, и теперь платформу Joomla нетрудно установить и использовать, не устанавливая CMS. У данной платформы имеется собственное хранилище исходного кода, доступное на веб-сайте по адресу `https://github.com/joomla/joomla-platform`. Оно может быть загружено отдельно.

Причины выделения платформы Joomla в отдельный проект

Для официального отделения платформы Joomla от проекта CMS имеются две главные причины. Первая из них — сделать проект более привлекательным для разработчиков. Система Joomla CMS стала вполне зрелым, самостоятельным проектом с миллионами пользователей. И хотя для ее усовершенствования предстоит еще немало сделать, уже в 2005 году она отделилась от проекта Mambo, начатого в 2000 году. Таким образом, CMS находится на той стадии, когда большинство вносимых в нее изменений носят поэтапный характер. Более того, вся отрасль CMS достигла в настоящее время относительно зрелой стадии развития, хотя бы по сравнению со многими другими видами веб-приложений.

С другой стороны, платформа Joomla предоставляет разработчикам уникальную возможность. Ведь это новый проект, в котором разработчики могут принять участие, работая над новыми интересными функциональными возможностями, не испытывая никаких ограничений со стороны уже имеющегося кода. У них имеется немало возможностей для написания нового кода и разработки функциональных возможностей с самого начала.

А с другой стороны, данная платформа уже применяется на миллионах веб-сайтов Joomla и поэтому как следует проверена на практике и заслуживает полного доверия. Большинство новых функциональных возможностей на платформе окажутся со временем доступными и для всего базового контингента пользователей системы Joomla CMS, как только новые средства платформы станут выгодно использоваться в CMS. Это означает, что разработчики, дополняющие платформы важными средствами, сделают их потенциально доступными для миллионов пользователей.

Другая главная причина выделения платформы в отдельный проект — стремление упростить применение данной платформы разработчиками в других веб-приложениях, кроме тех, что основаны на CMS. Несмотря на всю важность управления информационным наполнением, это всего лишь одна из возможных разновидностей веб-приложений. Ныне веб служит самым разным целям, помимо веб-сайтов для обмена информацией, и поэтому весьма актуальной становится разработка приложений в самых передовых областях применения веб.

Для большинства веб-приложений требуются самые основные функциональные возможности, в том числе аутентификация пользователей, взаимодействие с файловой системой, фильтрация данных и прочее. Когда группа разработчиков приступает к работе над новым веб-приложением, вместо разработки его функциональных возможностей собственными силами они могут воспользоваться платформой Joomla, чтобы положить неплохое начало своему проекту. Разворачивая программные средства, которые уже содержат методы, удовлетворяющие большинству общих требований к веб-приложению, разработчики могут уделить основное внимание тем частям проекта, которыми он отличается от других аналогичных проектов, не тратя зря время на "изобретение колеса".

Назначение платфомы Joomla

Ныне многие виды приложений выполняются непосредственно в веб. Ниже перечислены лишь некоторые их примеры.

- Программы электронной коммерции, предназначенные для обработки данных в фоновом режиме на любых веб-сайтах, где можно совершать закупки.
- Веб-ориентированные системы планирования ресурсов предприятий, предназначенные для финансового учета и связанных с ним функций на самых разных предприятиях и в организациях.
- Приложения, допускающие видоизменение веб-сайта в результате запуска программы на выполнение из командной строки на консоли сервера.
- Форумы пользователей и электронные доски объявлений.

Платформа Joomla может стать прочным основанием для разработки большинства разновидностей веб-приложений. Ее можно, в частности, рассматривать в качестве неплохой отправной точки для внедрения в разрабатываемое приложение любых функциональных возможностей, встроенных в данную платформу.

Примеры разработки программ на платформе Joomla

Рассмотреть особенности платформы Joomla лучше всего на конкретных примерах. В хранилище исходного кода можно найти немало примеров программ, разработанных на данной платформе, обратившись по адресу `https://github.com/joomla/joomla-platform-examples`. Начнем с двух простейших примеров. В одном из них используется браузер, а в другом — системный интерфейс командной строки. Программы с интерфейсом командной строки запускаются на выполнение из командной строки на консоли, а не в окне браузера.

Подготовка платформы проекта

Прежде всего необходимо загрузить файлы платформы Joomla и подготовить соответствующую структуру папок. Для этого достаточно выполнить следующие действия.

1. Создайте папку для хранения файлов проекта, присвоив ей имя `platform-test`. Если требуется проверить платформу как в приложениях для браузера, так и в приложениях командной строки, разместите эту папку в папке `DocumentRoot` на веб-сервере (например, в папке `htdocs`).
2. Загрузите платформу. Для этого достаточно перейти по адресу `https://github.com/joomla/joomla-platform` и щелкнуть на кнопке Tags (Метки). В итоге появится список официально выпущенных версий данной платформы (например, ссылка на архивный файл `11.4.zip`). С другой стороны, вы можете загрузить последнюю версию хранилища исходного кода, щелкнув последовательно на кнопках Downloads (Загрузки) и Download as zip (Загрузить как архив формата zip).
3. Распакуйте архив в папку `platform-test`. Папка самого верхнего уровня в структуре папок архива называется так же, как и сам архив, т.е. ей присваивается имя в следующем формате: `joomla-joomla-platform-<версия>-<номер массива изменений>` (например, `joomla-joomla-platform-11.4-0-g4329ba0.zip`). После распаковки архива переименуйте эту папку в `joomla-platform`. Таким образом, в папке `platform-test` должна оказаться подпапка `joomla-platform`, а в

ней — подпапки build, docs, libraries, media и tests. Следует, однако, иметь в виду, что для нормальной работы данной платформы потребуются только папки libraries и media, тогда как остальные папки нужны исключительно для разработки программ.

4. Загрузите примеры программ. Для этого перейдите по адресу https://github.com/joomla/joomla-platform-examples и щелкните последовательно на кнопках Downloads и Download as zip. В итоге будет загружен архивный файл вроде joomla-joomla-platform-examples-544306f.zip.
5. Распакуйте архив во временную папку и скопируйте папки cli и web в папку platform-test. Кроме того, скопируйте в эту же папку файл bootstrap.dist.php.
6. Таким образом, в папке platform-test должны оказаться три подпапки: cli, joomla-platform и web, а также файл bootstrap.dist.php.
7. Скопируйте файл bootstrap.dist.php в файл bootstrap.php. Отредактируйте этот файл, введя в него следующую строку кода:

   ```
   require dirname(__FILE__).'/joomla-platform/libraries/import.php';
   ```

   В этой строке кода платформа уведомляется о том, где следует искать файл со сценарием импорта. Для этой цели вызывается функция dirname() языка PHP, получающая сначала имя каталога, в котором находится текущий файл (в данном случае — bootstrap.php), а затем использующая это имя для составления полного пути к файлу со сценарием импорта. В данном примере файл bootstrap.php оказывается в той же папке, что и папка joomla-platform.

Теперь вы должны быть в состоянии запустить на выполнение все примеры программ. Для проверки такой возможности попробуйте выполнить пример программы Hello World, выполнив следующие действия.

- Начните сеанс работы в режиме командной строки на своем компьютере и перейдите к каталогу platform-test/cli/101-hello-world.
- Убедитесь в том, что проверяемая программа на PHP запускается на выполнение из командной строки. Если требуется, введите ее в путь к исполняемому файлу.
- Введите в командной строке следующую команду:

  php run.php
- На консоли должен появиться следующий результат выполнения проверяемой программы:

  Hello World!

Если вам не удастся добиться этого результата, проверьте правильность синтаксиса команды require в файле bootstrap.php. По этой команде загружается файл начальной загрузки.

Кроме того, вы можете выполнять веб-приложения, если папка platform-test находится в папке DocumentRoot на веб-сервере. Например, для того чтобы выполнить веб-приложение detect-client, откройте окно браузера и введите приведенный ниже URL в поле адреса.

```
<путь к папке platform-test>/web/detect-client/index.php
```

Так, если папка platform-test находится в папке htdocs на локальной машине, то упомянутый выше URL будет выглядеть следующим образом:
http://localhost/platform-test/web/detect-client/index.php

В результате выполнения данного веб-приложения в окне браузера должно появиться нечто, аналогичное следующему:

```
Welcome to the Joomla! Platform's JWeb class.

    * User-agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US;
↳rv:1.9.2.23) Gecko/20110920 Firefox/3.6.23 ( .NET CLR 3.5.30729;
↳.NET4.0C)
    * Is a mobile device? No
    * Platform: 1
    * Engine: 13
    * Browser: 18 (3.6.23)
```

Если и в этом случае вы получите сообщение об ошибке, проверьте правильность содержимого файла bootstrap.php.

Приложение командной строки Hello World

Рассмотрим далее исходный код приложения командной строки Hello World из файла platform-test/joomla-platform/cli/101-hello-world/run.php). Ниже приведена первая часть исходного кода этого приложения.

```
// Это достоверная точка входа в Joomla!
// Требуется загрузка платформы Joomla! из файла import.php.
define('_JEXEC', 1);

// задать константу, связанную с базовым путем к файлам
// Это одна из немногих констант, обязательных для платформы Joomla!
define('JPATH_BASE', dirname(__FILE__));

// выполнить начальную загрузку приложения
require dirname(dirname(dirname(__FILE__))).'/bootstrap.php';

// импортировать класс JCli с платформы
jimport('joomla.application.cli');
```

В этой части исходного кода сначала определяется константа _JEXEC, уведомляющая последующие программные файлы о том, что они выполняются в приложении Joomla. Эта константа требуется для выполнения файлов рассматриваемой здесь платформы. Затем в данной части кода определяется константа JPATH_BASE. Это нужно сделать обязательно, хотя ее применение зависит от конкретного приложения. Как правило, она задает путь к корневой папке приложения. В данном случае константа JPATH_BASE не используется, и поэтому в ней задается путь к текущей папке.

Далее следует команда require, включающая отредактированный ранее файл bootstrap.php. В данном случае включаемый файл находится в корневой папке (platform_test), которая находится на три уровня иерархии выше текущей папки (platform_test/cli/101-hello-world), и поэтому для доступа к ней выполняются подряд три команды dirname(). Именно такая структура папок используется в данном

примере. Файл bootstrap.php загружает, в свою очередь, файл libraries/import.php, в котором определяется целый ряд констант для данной платформы и загружаются некоторые из основных ее классов.

И последняя команда в данной части кода импортирует файл cli.php из папки libraries/joomla/application. В этом файле объявляется класс, который обычно расширяют все приложения командной строки на данной платформе.

Ниже приведена остальная часть исходного кода из примера приложения Hello World.

```
class HelloWorld extends JCli
{
  /**
   * Выполняет приложение
   *
   * Метод 'execute' служит точкой входа в приложение командной строки.
   *
   * @return            void
   *
   * @since       11.3
  */
  public function execute()
  {
    // отправить символьную строку на стандартный вывод
    $this->out('Hello world!');
  }
}
// получить экземпляр объекта приложения, передав имя его класса
// методу JCli::getInstance(), а также воспользоваться связыванием
// методов в цепочку, чтобы выполнить приложение
JCli::getInstance('HelloWorld')->execute();
```

В этой части исходного кода рассматриваемого здесь примера приложения создается класс, расширяющий класс JCli. В этом классе определяется единственный метод execute(), в котором вызывается метод out() из класса JCli для вывода сообщения на консоль. После этого закрывается сначала объявление данного метода, а затем и его класса.

И в последней строке из данной части кода вызываемые методы связываются в цепочку для выполнения двух действий. Сначала получается новый объект типа JCli с помощью метода getInstance(), а затем для данного объекта вызывается метод execute(). В конечном итоге на консоль выводится сообщение «Hello world!» (Здравствуй, мир). Следует иметь в виду, что эта последняя строка кода оказывается за пределами объявления класса HelloWorld данного приложения, и поэтому она выполняется немедленно.

Веб-приложение Hello WWW

Это веб-приложение доступно в файле web/101-hello-www/index.php и выводит простое текстовое сообщение в окне браузера. Ниже приведена первая часть его исходного кода.

```
// Это достоверная точка входа в Joomla!
define('_JEXEC', 1);
```

```
// задать константу, связанную с базовым путем к файлам
define('JPATH_BASE', dirname(__FILE__));

// довести уведомление об ошибках до любого количества
// отображаемых ошибок, но эти установки не должны
// использоваться в окончательном варианте
error_reporting(E_ALL);
ini_set('display_errors', true);

// выполнить начальную загрузку приложения
require dirname(dirname(dirname(__FILE__))).'/bootstrap.php';

// импортировать класс JWeb с платформы
jimport('joomla.application.web');
```

Как и прежде, в первой части исходного кода данного приложения определяются константы. В двух последующих строках уведомление об ошибках доводится до максимально возможного. Это полезно на стадии разработки для отладки любого кода, нарушающего строгие нормы языка PHP. Следует также заметить, что данная платформа предназначена для выполнения программ на PHP, начиная с версии 5.3. И в последних двух строках сначала включается файл `bootstrap.php` из корневой папки проекта, а затем из библиотеки импортируется файл `libraries/application/web.php`, в котором объявляется класс JWeb.

Ниже приведена остальная часть исходного кода данного веб-приложения из файла `index.php`.

```
class HelloWww extends JWeb
{
  /**
   * Переопределяет родительский метод doExecute() для выполнения
   * веб-приложения. В этот метод должен быть включен специальный
   * код, выполняющий данное веб-приложение.
   *
   * @return void
   *
   * @since 11.3
  */
  protected function doExecute()
  {
    // Это приложение лишь выводит простой HTML-документ.
    // Для подготовки к выводу используется метод setBody().
    // Вывод всех заголовков и прочего возьмет на себя класс JWeb.

    $this->setBody('<html>
      <head>
        <title>Hello WWW</title>
      </head>
      <body style="font-family:verdana;">
        <p>Hello WWW!</p>
      </body>
      </html>'
```

```
    );
  }
}
// получить экземпляр объекта приложения, передав имя его класса
// методу JWeb::getInstance(), а также воспользоваться связыванием
// методов в цепочку, чтобы выполнить приложение
JWeb::getInstance('Hellowww')->execute();
```

Сначала в завершающей части исходного кода рассматриваемого здесь приложения создается класс `Hellowww`, расширяющий класс `JWeb`. Большинство веб-приложений, разрабатываемых на платформе Joomla, расширяют класс `JWeb`. Затем создается защищенный метод `doExecute()`, переопределяющий одноименный метод из класса `JWeb`. Напомним, что в примере приложения командной строки Hello World был создан аналогичный метод `execute()`. И тот и другой метод может быть использован в пределах своего класса, но у них имеется одно существенное отличие. Если проанализировать исходный код метода `execute()` из класса `JWeb`, то можно обнаружить, что он инициирует события до и после вызова метода `doExecute()`, а затем воспроизводит документ, инициирует ряд других событий и наконец посылает запрос клиенту, опять же инициируя события до и после отправки запроса. Все эти действия обычно требуются для обработки HTML-документов, и поэтому целесообразнее переопределить метод `doExecute()`, предоставив методу `execute()` сделать все остальное.

В исходном коде метода `doExecute()` из класса `JWeb` вызывается метод `setBody()`, подготавливающий выводимое текстовое сообщение к выводу в формате HTML-документа. В состав класса `JWeb` входят различные методы для обработки заголовка и основного текста документа.

Последняя строка кода данного приложения похожа на аналогичную строку из предыдущего примера приложения. В ней сначала получается экземпляр объекта, а затем вызывается метод `execute()`. В конечном итоге текстовое сообщение выводится в HTML-документе, отображаемом в окне браузера.

Пример текущего контроля подписок

А теперь, когда стали понятны основы применения платформы Joomla для разработки приложений, рассмотрим более практический пример, в котором используется компонент Joompro Subscriptions, созданный в предыдущих главах этой книги. Напомним, что этот компонент служит для создания подписок, срок действия которых истекает через определенное количество дней. Но на практике требуется каким-то образом контролировать эти подписки и вести их учет.

В данном примере будет создано приложение командной строки, выполняющее две функции. Во-первых, оно проверяет наличие в базе данных подписок, срок действия которых истекает в ближайшие пять дней, а затем посылает по электронной почте соответствующее уведомление подписчику. И во-вторых, оно удаляет из таблицы соответствий подписку, если срок ее действия истек, а пользователя исключает из группы подписок. Предполагается, что данное приложение будет выполняться по графику автоматически (например, по команде `cron` в ОС Linux). Именно поэтому результаты его выполнения будут записываться в журнал регистрации для дальнейшего просмотра администратором веб-сайта вместо вывода на консоль.

Структура проекта

Для целей данного примера будет выбрана несколько иная структура проекта, чем в рассмотренных ранее примерах. В частности, файлы программ и журналов регистрации должны размещаться в папке `src`, находящейся в папке `joomla-platform`. С этой целью создайте сначала новую папку `platform-test/joomla-platform/src`, а в ней — папку `logs` для хранения файлов с журналами регистрации. Таким образом, в папке `src` должны оказаться три файла и одна папка, как показано ниже.

- Файл `monitor.php`: программа, выполняемая из командной строки.
- Файл `subscriptionmonitor.php`: программа, контролирующая подписки.
- Файл `configuration.php`: содержит данные конкретной конфигурации веб-сайта.
- Папка `logs`: содержит создаваемые по ходу дела файлы журналов регистрации.

Конфигурационный файл

Первым необходимо создать конфигурационный файл. У него должна быть такая же структура, как и у конфигурационного файла системы Joomla CMS, и для этого достаточно скопировать файл `configuration.php` из корневой папки установленной среды Joomla. Для целей данного примера потребуются лишь те поля из структуры конфигурационного файла, которые используются для установления связи с базой данных CMS и отправки электронной почты, хотя в этот файл будут включены и другие поля. Ниже приведен исходный код для ввода в файл `configuration.php`.

```
<?php

// предотвратить прямой доступ к этому файлу за
// пределами вызывающего приложения
defined('_JEXEC') or die;
class JConfig
{
  public $dbtype        = 'mysqli';
  public $host          = '127.0.0.1';
  public $user          = 'test';
  public $password      = 'password';
  public $db            = 'book_170';
  public $dbprefix      = 'jos_';
  public $ftp_host      = '127.0.0.1';
  public $ftp_port      = '21';
  public $ftp_user      = '';
  public $ftp_pass      = '';
  public $ftp_root      = '';
  public $ftp_enable    = 0;
  public $tmp_path      = '/tmp';
  public $log_path      = '/var/logs';
  public $mailer        = 'smtp';
  public $mailfrom      = 'admin@ваш_сайт.com';
  public $fromname      = 'Joomla! Programming';
  public $sendmail      = '/usr/sbin/sendmail';
  public $smtpauth      = '1';
```

```
  public $smtpuser      = '<имя пользователя gmail>';
  public $smtppass      = '<пароль gmail>';
  public $smtphost      = 'smtp.gmail.com';
  public $smtpsecure    = 'ssl';
  public $smtpport      = '465';
  public $debug         = 0;
  public $caching       = '0';
  public $cachetime     = '900';
  public $language      = 'en-GB';
  public $secret        = null;
  public $editor        = 'none';
  public $offset        = 0;
  public $lifetime      = 15;
}
```

Установки в этом конфигурационном файле вам, скорее всего, придется изменить, чтобы согласовать их со своей программной средой. В частности, изменения должны быть внесены в поля `$user` (имя пользователя), `$password` (пароль доступа к базе данных), `$db` (имя базы данных), `$dbprefix` (префикс таблицы базы данных), а возможно, и в установки электронной почты (в зависимости от конкретной системы). По желанию вы можете выполнить код из данного примера и без электронной почты. Установки электронной почты приведены выше для учетной записи пользователя бесплатной почты Google Gmail.

Контрольный файл

Этот файл служит в качестве точки входа в рассматриваемое здесь приложение. Его имя указывается как команда для выполнения из командной строки. Ниже приведена первая часть исходного кода из этого файла.

```
<?php
// объявить, что это достоверная точка входа в Joomla!,
// если еще не объявлено
if (!defined('_JEXEC'))
{
        define('_JEXEC', 1);
}
// задать константу для базового пути к файлам приложения,
// если она еще не задана
if (!defined('JPATH_BASE'))
{
        define('JPATH_BASE', dirname(__FILE__));
}
// импортировать платформу Joomla!
require dirname(dirname(__FILE__)) . '/libraries/import.php';
// импортировать библиотечные зависимости
jimport('joomla.log.log');
require JPATH_BASE . '/subscriptionmonitor.php';
```

В этой части исходного кода из контрольного файла проверяется, определены ли константы `_JEXEC` и `JPATH_BASE`. Следует иметь в виду, что в данном примере файл начальной загрузки `bootstrap.php` не используется, а вместо него включается файл импорта.

Кроме того, команда `require` настраивается на поиск файла `import.php` относительно местоположения текущего файла (т.е. контрольного файла `monitor.php`). Как только файл `import.php` будет загружен, из библиотеки могут быть импортированы все классы, необходимые для нормальной работы данного приложения. Для этой цели требуется только класс `JLog`. Затем включается программный файл `subscriptionmonitor.php`.

Ниже приведена следующая часть исходного кода из контрольного файла.

```
// загрузить конфигурационный файл configuration.php
$config = JFactory::getConfig(JPATH_BASE.'/configuration.php');

// создать журнал регистрации
// получить дату, чтобы просматривать журналы регистрации
// в определенном временном интервале
$date = JFactory::getDate()->format('Y-m-d');

// добавить регистратор
JLog::addLogger(
  // передать массив параметров конфигурации
  // Примечание: по умолчанию в файл журнала регистрации
  // записывается отформатированный текст
  array(
      // задать имя файла для журнала регистрации
      'text_file' => 'monitor-.'.$date.'.php',
      // задать путь к файлам с журналами регистрации
      'text_file_path' => __DIR__.'/logs/'
      )
);
```

В этой части исходного кода устанавливается конфигурация с помощью созданного ранее файла `configuration.php`. Следует заметить, что для этой цели вместо объекта в переменной `$config` при выполнении метода `getConfig()` создается объект типа `JConfig` (`JFactory::$config`), который используется сначала в методе `JFactory::getDate()`, а затем и в методе `JFactory::getMailer()`. Благодаря этому в самом начале данного приложения задается нужный конфигурационный файл `configuration.php`. Если же, например, вызвать метод `JFactory::getDate()` до создания объекта типа `JConfig`, то загружен будет файл `configuration.php` из места, задаваемого по умолчанию с помощью константы `JPATH_PLATFORM`, а это совсем не то, что нам требуется.

Далее файл журнала регистрации создается средствами класса `JLog`. В его имя включается текущая дата, и благодаря этому каждый день получается новый файл. Этот файл размещается в созданной ранее папке `logs`.

Ниже приведена следующая часть исходного кода из контрольного файла.

```
// заключить выполнение кода в блок операторов try/catch,
// чтобы перехватить любые исключения, генерируемые в сценарии
try
{
    // получить экземпляр объекта и выполнить приложение
    JCli::getInstance('SubscriptionMonitor')->execute();
}
catch (Exception $e)
{
```

```
    // исключение перехвачено, добавить сообщение в журнал регистрации
    JLog::add($e->getMessage(), JLog::ERROR);
    exit($e->getCode());
}
```

В этой части исходного кода создается блок операторов `try/catch`, в котором сначала получается экземпляр объекта типа `JCli`, а затем выполняется метод `execute()` из класса `SubscriptionMonitor`. Как будет показано далее, в этом классе имеется целый ряд методов, генерирующих исключения, поэтому если метод `execute()` вызывается внутри блока оператора `try`, то этим гарантируется перехват и обработка любых исключений, генерируемых в каком угодно месте этого метода. А в блоке оператора `catch` сообщение о возникшей исключительной ситуации записывается в журнал регистрации. Следует иметь в виду, что данное приложение предназначено для автоматического выполнения на сервере, поэтому возникающие в нем ошибки лучше записывать в журнал регистрации, чем выводить на консоль.

Весь код текущего контроля подписок можно было бы поместить в одном этом контрольном файле, чтобы не создавать еще один. Принятый в данном случае подход имеет следующее важное преимущество: он позволяет очень легко вводить дополнительные функции текущего контроля. Допустим, требуется также контролировать дату окончания публикации статей. Для этого достаточно создать новый класс текущего контроля статей и добавить строку кода для его выполнения в блок оператора `try` из файла `monitor.php`. В одном этом файле можно было бы организовать функции текущего контроля всего вебсайта, назначив его выполнение на сервере в виде одного простого задания по команде `cron`.

Файл текущего контроля подписок

Этот файл, называемый `subscriptionmonitor.php`, по существу, выполняет функции проверки и обновления базы данных, а также отправки сообщений по электронной почте. Ниже приведена первая часть исходного кода из этого файла.

```
jimport('joomla.application.cli');
jimport('joomla.database.database');

/**
 * Приложение на платформе Joomla! для текущего контроля
 * состояния подписок и выполнения необходимых действий.
 *
 * @package Subscriptions
 * @since   1.0
*/
class SubscriptionMonitor extends JCli
{
    /**
     * @var       JDatabase         Объект для установления связи
     *                              приложения с базой данных.
     * @since     1.0
     */
     protected $db;
```

Сначала в этой части исходного кода импортируются классы JCli и JDatabase, а затем объявляется класс SubscriptionMonitor в качестве подкласса, производного от класса JCli. И наконец, в этот класс вводится защищенное поле $db, в котором должен храниться объект типа JDatabase для данного класса.

Ниже приведена следующая часть исходного кода из файла subscriptionmonitor.php.

```
public function__construct(JInputCli $input = null, JRegistry $config =
↳null, JDispatcher $dispatcher = null, JDatabase $db = null)
{
    // вызвать конструктор родительского класса для
    // основной установки объекта
    parent::__construct($input, $config, $dispatcher);

    // воспользоваться объектом для связи с базой данных,
    // если он задан
    if ($db instanceof JDatabase)
    {
        $this->db = $db;
    }
    // установить соединение с базой данных, исходя из логики
    // выполнения данного приложения
    else
    {
        $this->loadDatabase();
    }
}
```

В сигнатуре вызова конструктора из приведенного выше кода присутствуют следующие четыре аргумента: $input, $config, $dispatcher и $db. И хотя в данном примере эти аргументы не требуются, они все же указаны на тот случай, если они понадобятся при использовании данного класса в другом контексте. В частности, первый аргумент позволяет передать данному конструктору объект типа JInputCli, имитирующий аргументы командной строки. А в качестве остальных аргументов при вызове данного конструктора можно указать собственный конфигурационный файл, класс диспетчера и класс базы данных.

Обратите также внимание на еще одно отличие в сигнатуре вызова данного конструктора по сравнению с тем, что встречалось в предыдущих примерах исходного кода. В данном случае перед каждым аргументом явно указывается класс (например, JInputCli $input). Это языковое средство PHP называется *явным указанием типов*. Оно явным образом сообщает интерпретатору PHP (и, конечно, тому, что читает исходный код) о том, что данный аргумент должен быть объектом указанного типа. В противном случае при выполнении метода или конструктора возникнет неисправимая ошибка. Однако явное указание типов стало доступным лишь с версии PHP 5.3. В коде данного конструктора сначала вызывается конструктор родительского класса, а затем проверяется, установлено ли поле $db. Если этого не сделано, то для установки данного поля вызывается метод loadDatabase().

Далее в классе SubscriptionMonitor определяется метод doExecute(). Напомним, что этот метод вызывается в методе execute() из класса JCli. Ниже приведен исходный код этого метода.

```
protected function doExecute()
{
    // зарегистрировать начало программы
    JLog::add('Subscription monitoring started');

    // получить список окончившихся подписок и удалить их
    $ended = $this->getEndedSubscriptions();
    foreach ($ended as $sub)
    {
        $this->doEndSubscription($sub);
    }

    // получить список завершающихся подписок и послать уведомление
    $ending = $this->getEndingSubscriptions();
    foreach ($ending as $sub)
    {
        $this->doSendSubscriptionEndingNotification($sub);
    }

    // зарегистрировать конец программы
    JLog::add('Subscription monitoring ended');
}
```

Именно в этом методе фактически осуществляется управление выполнением данного приложения. Сначала в нем делается запись в журнал регистрации о начале выполнения программы. Затем вызывается метод getEndedSubscriptions() для получения списка подписок, срок действия которых уже истек. Все они просматриваются в цикле, и для обработки каждой устаревшей подписки вызывается метод doEndSubscription(). Что касается применения циклов foreach в данном контексте, то следует обратить внимание на одну интересную особенность. Вполне возможно, что устаревшие подписки не обнаружатся, и тогда массив $ended окажется пустым. В таком случае цикл foreach будет просто пропущен без каких-либо ошибок или замечаний.

После обработки устаревших подписок вызывается метод getEndingSubscriptions() для получения списка подписок, срок действия которых близок к истечению. И в этом случае каждая подписка обрабатывается в цикле foreach, для чего вызывается метод doSendSubscriptionEndingNotification(). И в конце рассматриваемого здесь метода в журнале регистрации делается запись о завершении программы.

Далее в классе SubscriptionMonitor определяется метод getEndedSubscriptions(). В этом методе делается запрос базы данных на получение списка устаревших подписок. Ниже приведен исходный код этого метода:

```
protected function getEndedSubscriptions()
{
  // получить объект текущей даты
  $now = JFactory::getDate($this->get('execution.datetime'));

  // получить класс для составления запроса из базы данных
```

```
  $query = $this->db->getQuery(true);

  // составить запрос на выборку подписок с датой окончания,
  // наступившей раньше текущей даты
  $query->select('*')
        ->from($this->db->qn('#__joompro_sub_mapping'))
        ->where($this->db->qn('end_date') . ' < ' .
↳$this->db->q($now->toSQL()))
        ->where($this->db->qn('end_date') . ' > ' .
↳$this->db->q('0000-00-00 00:00:00'));

  // установить объект запроса базы данных в качестве
  // объекта для связи с базой данных
  $this->db->setQuery($query);

  // получить все строки, возвращаемые из таблицы по запросу,
  // в виде массива объектов
  $rows = $this->db->loadObjectList();

  return $rows;
}
```

Сначала в данном методе получаются текущие дата и время, а затем составляется запрос на выборку всех строк из таблицы соответствий, в столбце `end_date` которых хранятся более ранние дата и время, чем текущие. Для получения текущих даты и времени служит приведенный ниже код.

```
($this->get('execution.datetime')
```

В этом коде вызывается метод `get()` из класса `JCli`. Ниже приведен исходный код данного метода.

```
public function get($key, $default = null)
{
  return $this->config->get($key, $default);
}
```

В коде данного метода вызывается метод `get()` для поля `config` текущего класса. А где же устанавливается значение `execution.datetime` для текущих даты и времени? Это делается в конструкторе класса `JCli`, где имеется следующая строка кода:

```
$this->set('execution.datetime', gmdate('Y-m-d H:i:s'));
```

Результат выполнения метода `gmdate()`, вызываемого в методе `set()`, вводится в поле `config` текущего класса. Метод `gmdate()` возвращает текущее время по Гринвичу. А зачем сохранять это значение в поле `config` вместо того, чтобы вызывать метод `gmdate()` всякий раз, когда требуется текущее время? Дело в том, что при каждом вызове метода `gmdate()` из него будут возвращаться разные значения. Если же вызвать его один раз в конструкторе, то будет получено одно значение текущего времени, используемое в запросах, а следовательно, они будут всегда давать непротиворечивые результаты.

Кроме того, дата окончания подписки не должна быть нулевой, что проверяется специально. Напомним, что в системе Joomla CMS нулевой датой принято обозначать неопределенный срок действия. Добавление проверки на нулевую дату дает возможность применять такое же условное обозначение срока действия и для подписок.

Для составления запроса вызываемые методы связываются в цепочку. благодаря этому весь запрос формируется в одной строке кода, где результаты выполнения предыдущего метода передаются в качестве объекта следующему вызываемому методу. В противном случае методы `select()`, `from()` и дважды метод `where()` пришлось бы вызывать в отдельных строках кода. Как только запрос будет составлен, он передается объекту базы данных, а результаты его обработки возвращаются из метода `loadObjectList()`. Напомним, что этот метод возвращает массив объектов.

И наконец, в коде рассматриваемого здесь метода обращает на себя внимание еще одна интересная особенность. В нескольких местах этого кода вызывается метод `qn()` из класса `JDatabase`. Но если проанализировать исходный код объявления этого класса в файле `libraries/joomla/database/database.php`, то в нем не удастся обнаружить метод под названием `qn()`. А поскольку класс `JDatabase` не расширяет ни один из родительских классов, то где же находится определение метода `qn()`? Разгадка кроется в методе `__call()` из класса `JDatabase`. Это один из нескольких "волшебных" методов языка PHP. Обращение к нему происходит всякий раз, когда вызывается метод, неопределенный в текущем классе. Чаще всего метод `__call()` применяется для создания псевдонимов наиболее употребительных методов. Ниже приведен исходный код этого метода.

```
switch ($method)
{
  case 'q':
  return $this->quote ($args[0], isset($args[1]) ? $args[1] : true);
  break;
  case 'nq':
  case 'qn':
    return $this->quoteName ($args[0]);
  break;
}
```

В этом методе по очереди проверяются имена методов `q`, `nq` и `qn` и затем вызывается метод `quote()` или `quoteName()`. Таким образом, метод `qn()` оказывается всего лишь псевдонимом метода `quoteName()`. Остается лишь обратить внимание на то, что при передаче аргументов (элементов массива `$args`) методам `quote()` и `quoteName()` необходимо соблюдать особую аккуратность.

Следующим в классе `SubscriptionMonitor` определяется метод `getEndingSubscriptions()`. Он подобен предыдущему методу, за исключением того, что в нем выбираются подписки, срок действия которых истекает в ближайшие пять дней. Ниже приведен исходный код этого метода.

```
protected function getEndingSubscriptions()
{
    // получить объекты двух дат: текущей даты и
    // наступающей через несколько дней
    $now = JFactory::getDate($this->get('execution.datetime'));
    $future = JFactory::getDate($this->get('execution.datetime'));
    $future->add(new DateInterval('P5D'));

    // получить класс для составления запроса из базы данных
    $query = $this->db->getQuery(true);
```

```
    // составить запрос на выборку подписок, срок действия которых
    // истекает через пять дней от текущей даты
    $query->select('*')
          ->from($this->db->qn('#__joompro_sub_mapping'))
          ->where($this->db->qn('end_date') . ' < ' .
↳$this->db->q($future->toSQL()))
          ->where($this->db->qn('end_date') . ' > ' .
↳$this->db->q($now->toMySQL()));

    // установить объект запроса базы данных в качестве
    // объекта для связи с базой данных
    $this->db->setQuery($query);

    // получить все строки, возвращаемые из таблицы по запросу,
    // в виде массива объектов
    $rows = $this->db->loadObjectList();
    return $rows;
}
```

Прежде всего в этом методе рассчитываются дата и время, наступающие в ближайшие несколько дней. Для этого создается объект текущей даты, сохраняемый в переменной `$future`, а затем к этой дате прибавляются пять дней с помощью метода `add()`. Для получения пятидневного промежутка времени используется класс `DateInterval` языка PHP, конструктор которого принимает аргумент в следующем формате:

```
P + <число> + <промежуток времени>
```

В данном примере в качестве аргумента указывается значение `P5D`, обозначающее промежуток времени в пять дней. В конечном итоге в переменной `$future` сохраняется значение даты и времени на пять дней позже, чем значение текущей даты и времени в переменной `$now`. Эти значения даты и времени затем используются для составления запроса на выборку любых подписок с датой окончания, наступающей позже текущей, но раньше, чем через пять дней. Как и прежде, все эти значения возвращаются в виде массива объектов.

Следует, однако, заметить, что составление такого запроса таит в себе потенциальные трудности для разработки. Если рассматриваемое здесь приложение должно выполняться ежедневно, то подписчик будет получать уведомление по электронной почте ежедневно в течение пяти дней, что не совсем приятно. Во избежание этого приложение текущего контроля подписок можно было бы выполнять реже (например, через каждые пять дней) или же внести изменения в запрос, чтобы выбирать подписки, оканчивающиеся в более узком временном промежутке (от четырех до пяти дней). С другой стороны, можно было бы записать в базе данных, что напоминающие уведомления уже отправлены по электронной почте, а затем использовать этот критерий для просеивания результатов запроса.

Следующим в классе `SubscriptionMonitor` определяется метод `doEndSubscriptions()`, выполняющий несколько функций.

- Получает название подписки и имя пользователя для отправки уведомления по электронной почте.
- Удаляет строку из таблицы соответствий пользователей и их групп подписки, в результате чего пользователь исключается из группы.

- Посылает по электронной почте уведомление пользователю о том, что подписка удалена.

Ниже приведена первая часть исходного кода данного метода.

```
protected function doEndSubscription($subscriptionMapping)
{
    $subscription = $this->getSubscription(
↳$subscriptionMapping->subscription_id);
   $user = $this->getUser($subscriptionMapping->user_id);
```

В этой части кода рассматриваемого здесь метода вызываются методы `getSubscription()` и `getUser()`. В них проверяется достоверность идентификаторов пользователя и подписки, а также получаются имя пользователя и название подписки для уведомления по электронной почте.

Ниже приведена следующая часть исходного кода из метода `doEndSubscriptions()`.

```
// составить запрос на исключение пользователя из группы
$query = $this->db->getQuery(true);
$query->delete()
      ->from($this->db->qn('#__user_usergroup_map'))
      ->where($this->db->qn('group_id') . ' = ' . (int)
↳$subscription->group_id)
      ->where($this->db->qn('user_id') . ' = ' .
↳ (int) $subscriptionMapping->user_id);

// установить объект запроса базы данных в качестве
// объекта для связи с базой данных
$this->db->setQuery($query);

// выполнить запрос
$this->db->query();
```

Итак, из таблицы соответствий известно, что подписка и пользователь достоверны, поэтому можно смело приступать к обновлению базы данных. С этой целью сначала составляется новый запрос на удаление строки из таблицы соответствий пользователей и их групп подписки, а затем этот запрос выполняется. Напомним, что пользователь был введен в группу, когда он подписывался на конкретную подписку.

Ниже приведена следующая часть исходного кода из рассматриваемого здесь метода.

```
// получить класс для составления запроса из базы данных
$query = $this->db->getQuery(true);
// составить зарос на удаление строки из таблицы соответствий
// пользователей и подписок
$query->delete()
      ->from($this->db->qn('#__joompro_sub_mapping'))
      ->where($this->db->qn('subscription_id') . ' = ' .
↳(int) $subscriptionMapping->subscription_id)
      ->where($this->db->qn('user_id') . ' = ' .
↳(int) $subscriptionMapping->user_id);

// установить объект запроса базы данных в качестве
```

```
// объекта для связи с базой данных
$this->db->setQuery($query);

// выполнить запрос
$this->db->query();
```

В этой части кода составляется новый запрос на удаление — на этот раз строки из таблицы соответствий пользователей и подписок. В итоге данная подписка пользователя отменяется.

И наконец, ниже приведена последняя часть исходного кода из метода `doEndSubscriptions()`.

```
// составить уведомление об окончании срока действия подписки
// для отправки пользователю по электронной почте
$subject = 'Your subscription has expired.';
$body[] = 'Hi, '.$user->name.',';
$body[] = '';
$body[] = 'Your subscription to '.$subscription->title.' has
➥expired.';
        $body[] = '';
        $body[] = 'Regards,';
        $body[] = $this->get('fromname');

        // отправить уведомление по электронной почте
        $this->sendNotificationEmail($user->email, $subject,
➥implode("\n", $body));
        JLog::add('Subscription removed for user='.$user->name.',
➥title='.$subscription->title);
}
```

В этой части исходного кода сначала формируется содержимое уведомления, отправляемого пользователю по электронной почте. Затем вызывается метод `sendNotificationEmail()` для отправки составленного уведомления по электронной почте. И наконец, в журнале регистрации делается запись о том, что подписка данного пользователя удалена.

Далее в рассматриваемом здесь классе следует определение метода `doSendSubscriptionEndingNotification()`. Его исходный код приведен ниже.

```
protected function doSendSubscriptionEndingNotification($subscriptionMapping)
  {
      $subscription = $this->getSubscription(
➥$subscriptionMapping->subscription_id);
$user = $this->getUser($subscriptionMapping->user_id);

      $subject = 'Your Subscription is ending soon.';
      $body[] = 'Hi, '.$user->name.',';
      $body[] = '';
      $body[] = 'Your subscription to '.$subscription- >title.
➥' will end on '.$subscriptionMapping->end_date.'.';
      $body[] = '';
      $body[] = 'Regards,';
      $body[] = $this->get('fromname');

      // отправить уведомление по электронной почте
```

```
        $this->sendNotificationEmail($user->email, $subject, implode("\n",
↳$body));

        // зарегистрировать уведомление в журнале
        JLog::add('Subscription ending notification sent for user='.
↳$user->name.', title='.$subscription->title);
    }
```

И этот метод начинается с получения объектов пользователя и подписки. Затем уведомление составляется и отправляется по электронной почте с помощью метода `sendNotificationEmail()`. И наконец, в журнале регистрации делается запись об отправке уведомления пользователю. Но в данном случае никаких изменений в базе данных не происходит, а лишь посылается уведомление, напоминающее пользователю о скором окончании его подписки.

Ниже приведен код метода `getSubscription()`, определяемого далее в классе `SubscriptionMonitor`.

```
protected function getSubscription($subID)
{
    // составить запрос на получение идентификатора группы
    // из записи о подписке в базе данных
    $query = $this->db->getQuery(true);
    $query->select('*')
          ->from($this->db->qn('#__joompro_subscriptions'))
          ->where($this->db->qn('id') . ' = ' . (int) $subID);

    // установить объект запроса базы данных в качестве
    // объекта для связи с базой данных
    $this->db->setQuery($query);

    // проверить наличие подписки
    $subscription = $this->db->loadObject();
    if (empty($subscription))
    {
        throw new Exception('Invalid Subscription.');
    }
    return $subscription;
}
```

В этом методе формируется запрос на получение строки из таблицы подписок по идентификатору подписки. Напомним, что этот идентификатор должен быть однозначным, а следовательно, по данному запросу из таблицы должна быть возвращена единственная строка. Если подписка отсутствует по данному идентификатору, генерируется исключение для регистрации этого события в журнале. И в конце рассматриваемого здесь метода возвращается объект `$subscription` с данными из полученной строки таблицы.

Далее в классе `SubscriptionMonitor` определяется метод `getUser()`. Ниже приведен исходный код этого метода.

```
protected function getUser($userID)
{
    // составить запрос на получение объекта пользователя
    $query = $this->db->getQuery(true);
    $query->select('*')
```

```
            ->from($this->db->qn('#__users'))
            ->where($this->db->qn('id') . ' = ' . (int) $userID);

        // установить объект запроса базы данных в качестве
        // объекта для связи с базой данных
        $this->db->setQuery($query);

        // проверить наличие пользователя
        $user = $this->db->loadObject();
        if (empty($user))
        {
            throw new Exception('Invalid User.');
        }
        return $user;
    }
```

Этот метод очень похож на предыдущий. В нем предпринимается попытка извлечь строку из таблицы пользователей по заданному идентификатору пользователя. Если это удастся сделать, то строка возвратится из таблицы в виде объекта. В противном случае генерируется исключение, указывающее на то, что идентификатор пользователя оказался недействительным.

Ниже приведен код метода `sendNotificationEmail()`, определяемого далее в классе `SubscriptionMonitor`.

```
protected function sendNotificationEmail($email, $subject, $body)
{
    JFactory::getMailer()->sendMail(
       $this->get('mailfrom'),
       $this->get('fromname'),
       $email,
       $subject,
       $body
    );
}
```

В этом методе отправляется уведомление по электронной почте, для чего из класса `JMail`, объявляемого в файле `libraries/joomla/mail/mail.php`, вызывается метод `sendMail()`. Однако веб-сайт должен быть специально настроен на отправку сообщений по электронной почте. Так, если вы работаете на локальной машине, не настроенной на обмен сообщениями по электронной почте, то можете оставить этот метод пустым, но по-прежнему определенным в данном классе, закомментировав все строки кода в его теле. Это позволит вам проверить на выполнение остальную часть рассматриваемого здесь приложения, даже если у вас не установлена электронная почта.

И последним в данном классе определяется метод `loadDatabase()`. Напомним, что этот метод вызывается в конструкторе текущего класса для создания объекта базы данных с информацией, извлекаемой из конфигурационного файла. Ниже приведен исходный код этого метода.

```
  protected function loadDatabase()
  {
    // если возникает ошибка при установлении соединения
    // с базой данных, то генерируется исключение
    $this->db = JDatabase::getInstance(
```

```
            array(
                'driver' => $this->get('dbtype'),
                'host' => $this->get('host'),
                'user' => $this->get('user'),
                'password' => $this->get('password'),
                'database' => $this->get('db'),
                'prefix' => $this->get('dbprefix'),
            )
        );
    }
} // конец класса
```

В этом методе просто создается новый объект типа `JDatabase`, а его поля заполняются значениями из конфигурационного файла. Как упоминалось выше, для выборки этих значений из конфигурационного файла вызывается метод `get()` из класса `JCli`.

Выполнение приложения для текущего контроля подписок

Для проверки приложения, рассматриваемого здесь в качестве примера, выполните следующие действия.

1. Введите одну или больше подписок в таблицу подписок.
2. Создайте пункт меню для отображения подписок.
3. Войдите в пользовательскую часть веб-сайта и подпишитесь на ряд подписок.
4. Измените дату окончания одной из подписок в таблице `os_joompro_sub_mapping` с помощью phpMyAdmin или другой программы, предназначенной для работы с базой данных, чтобы срок действия этой подписки оказался истекшим, а другой — истекающим раньше, чем через пять ближайших дней.
5. Убедитесь в том, что в файле `configuration.php` содержится правильная информация для соединения с базой данный веб-сайта.
6. Закомментируйте код в теле метода `sendNotificationEmail()` на тот случай, если электронная почта не установлена на той машине, где проверяется данное приложение.
7. Начните сеанс работы в режиме командной строки и перейдите к папке `platform-test/src`. Введите следующую команду:

   `php monitor.php`

Приложение должно запуститься на выполнение и завершиться возвратом к системному приглашению в командной строке. В папке `logs` должен появиться файл вновь созданного журнала регистрации с записями, аналогичными приведенным ниже.

```
#<?php die('Forbidden.'); ?>
#Date: 2012-10-15 01:07:35 UTC
#Software: Joomla! Platform 11.2.0 Stable [ Omar ] 27-Jul-2011 00:00 GMT

#Fields: datetime priority category message
2012-10-15T01:07:35+00:00 WARNING deprecated
↳JDatabaseMySQLi::hasUTF() is deprecated.
2012-10-15T01:07:35+00:00 INFO - Subscription monitoring started
```

```
2012-10-15T01:07:35+00:00 INFO - Subscription removed for
↳user=George Washington, title=Pontiac GTO
2012-10-15T01:07:35+00:00 INFO - Subscription ending
↳notification sent for user=George Washington, title=Ford Mustang
2012-10-15T01:07:35+00:00 INFO - Subscription monitoring ended
```

В первой записи, сделанной в журнале регистрации, предупреждается о том, что в коде приложения используется не рекомендованный к применению, устаревший метод `JDatabaseMySQLi::hasUTF()`. Этим предупреждением можно пренебречь. В следующей строке из журнала регистрации уведомляется о начале выполнения программы текущего контроля подписок. Далее следуют строки, в которых сообщается об удалении одной подписки и отправке соответствующего уведомления. И в последней строке из журнала регистрации уведомляется о завершении данной программы.

Проверьте различные условия возникновения ошибок, введя недостоверные данные в таблицы базы данных или сымитировав неудачное завершение обработки запросов. В частности, измените идентификатор пользователя или подписки в таблице соответствия подписок, чтобы он не совпадал с существующим идентификатором пользователя или подписки, или же измените имя таблицы на недействительное. В обоих случаях сообщения об ошибках должны появиться в журнале регистрации.

Выполнение программ командной строки в системе Joomla CMS

Если требуется написать программы командной строки, взаимодействующие с CMS, то для их размещения имеются две возможности. В рассмотренном выше примере было разработано совершенно отдельное приложение, опирающееся на собственную версию платформы Joomla и действующее независимо от программ CMS. А что касается системы CMS, то в этом приложении достаточно было знать лишь порядок установления связи с базой данных и программой электронной почты. Данное приложение могло бы благополучно работать на отдельном сервере, например, для текущего контроля подписок на любом числе веб-сайтов с системой Joomla CMS, для чего потребовалось бы только создать разные конфигурационные файлы. С другой стороны, созданные приложения командной строки можно расположить и в структуре папок Joomla CMS. Так, в версии 1.7 для этой цели в структуру папок Joomla CMS была введена новая папка `cli`.

Приложение из рассмотренного выше примера можно выполнить из экземпляра CMS, скопировав папку `logs` и три программных файла (`configuration.php`, `monitor.php` и `subscriptionmonitor.php`) из папки `src`, находящейся в папке проекта `joomla-platform`, в папку `cli` установленной системы Joomla CMS, а затем установив нужные значения в конфигурационном файле. После этого следует перейти к папке `cli` в системе Joomla CMS и запустить программный файл `monitor.php` на выполнение из командной строки. Данное приложение должно работать, как и прежде.

Следует также обратить внимание на сходство приемов программирования, использованных в данном примере и в предыдущих главах книги. Программирование на платформе Joomla мало чем отличается от программирования для системы CMS. После загрузки данной платформы для разработки приложений используются те же самые классы, средства доступа к базе данных и прочие приемы программирования, что и для приложений CMS.

Резюме

В этой главе были представлены основы разработки приложений на платформе Joomla. Сначала было показано, каким образом устанавливается данная платформа и как примеры разработанных на ней программ выполняются из командной строки и в веб-браузере. Попутно была проанализирована работа этих программ, в том числе порядок загрузки в них требующихся классов из библиотеки.

И наконец, на конкретном примере была продемонстрирована разработка реальной прикладной программы командной строки, вполне способной работать в автоматическом режиме на веб-сервере. Эта программа осуществляет текущий контроль подписок, срок действия которых уже истек или истекает. Пример данной программы наглядно показывает возможности платформы Joomla для разработки приложений, взаимодействующих с веб-сайтом на Joomla. Данная платформа представляет собой новый интересный проект с практически неограниченным потенциалом. Вероятнее всего, на ней будет разработано еще немало занимательных и полезных приложений. Программные средства, внедренные на платформе Joomla, позволяют также выгодно использовать преимущества системы Joomla CMS.

Приложение А

Краткий курс PHP и ООП

В этом приложении дается краткий курс объектно-ориентированного программирования (ООП) вообще и на языке PHP в частности. Пройдя этот курс, вы получите минимальные знания, которых требует программирование для Joomla!. Этот курс состоит из кратких, но полезных для практики пояснений, а для более углубленного изучения затрагиваемых в нем тем и вопросов следует обращаться к обширной литературе и ресурсам, доступным в Интернете.

Структура файлов PHP

PHP является интерпретируемым языком программирования, и поэтому файлы с расширением `.php`, содержащие исходный код программ на этом языке, могут быть отредактированы в любом текстовом редакторе. Следует, однако, иметь в виду, что для всех текстовых файлов в Joomla применяется кодировка UTF-8.

Файлы, содержащие только исходный код PHP

Большинство файлов PHP содержат только исходный код, написанный на языке PHP. Такие файлы начинаются с открывающего дескриптора `<?php`, после которого следует исходный код PHP до конца файла. Как правило, они *не* завершаются закрывающим дескриптором `?>`. Этот дескриптор является необязательным и на самом деле может вызвать сложности, если после него следует любое количество пробелов. Поэтому в файлах, содержащих только исходный код PHP, рекомендуется опускать закрывающий дескриптор PHP.

Файлы, содержащие исходный код PHP и HTML

Файлы компоновки и шаблонов `index.php` в Joomla состоят из определенного сочетания элементов разметки в коде PHP и HTML. В таких файлах исходный код

PHP указывается в дескрипторах PHP (<?php . . . ?>). Как поясняется далее в этом приложении, в файлах, содержащих смешанный код, обычно используется альтернативный синтаксис языка PHP.

Основы синтаксиса языка PHP

Пробелы (символы табуляции, пустой строки, возврата каретки, перевода строки и т.д.) игнорируются интерпретатором языка PHP, поэтому для удобочитаемости исходного кода очень важно отформатировать его правильно. Например, следующая строка кода:

```
$a=$b+$c;
```

равнозначна такой строке кода:

```
$a = $b + $c;
```

Но вторая строка более предпочтительна, поскольку ее легче читать. Кроме того, одна строка кода может занять несколько строк текста, как в приведенном ниже примере.

```
$a = $b + $c
   + $d;
```

Наиболее важные символы

Каждый оператор PHP должен завершаться точкой с запятой (`;`). Фигурные скобки (`{` и `}`) служат для заключения кода в блоки. Кодовые блоки предназначаются для того, чтобы начинать и оканчивать код в классах, функциях (называемых также методами), условных операторах `if`, операторах выбора `switch` и циклах. А круглые скобки служат для указания выражений в условных операторах `if`, операторах выбора `switch` и цикла **`while`**.

В строке кода, содержащей комментарии, после символов **`//`** могут быть введены любые символы. А комментарии, начинающиеся символами **`/*`** и оканчивающиеся символами **`*/`**, могут занимать несколько строк кода, как в приведенном ниже примере.

```
/* Это комментарий
   А это дополнительные комментарии
*/
// И это комментарий
$a = $b + $c; // а это еще один комментарий
```

Знаками тире и "больше" (**`->`**) обозначается вызов метода из класса. Например, в следующей строке кода:

```
$name = $db->loadResult();
```

вызывается метод `loadResult()` из объекта, хранящегося в переменной `$db`. А двойным двоеточием (`::`) обозначается вызов статических методов, как в приведенном ниже примере.

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

В табл. А.1 перечислены специальные символы, наиболее употребительные в языке PHP.

Таблица А.1. Специальные символы, употребляемые в языке PHP

Символ	Назначение
Точка с запятой (`;`)	Завершает оператор языка PHP
Фигурные скобки (`{` и `}`)	Заключают в блоки кода классов, функций или методов, условных операторов `if/then`, циклов и т.д.
Круглые скобки (`"(" и ")"`)	Предназначены для самых разных целей: заключают аргументы метода, логические и условные выражения, значения массива, операторы приведения типов
Двойная косая черта (`//`)	Обозначает однострочные комментарии, т.е. текст, вводимый в той же самой строке кода
Символы `/*` и `*/`	Обозначают блочные комментарии, т.е. весь текст, вводимый между этими символами в одной или нескольких строках кода
Знаки тире и "больше" (`->`)	Обозначают вызов функции или метода, например `$db->loadResult();`
Двойное двоеточие (`::`)	Обозначает вызов статического метода из класса, например `JFactory::getUser();`
Квадратные скобки (`[` и `]`)	Обозначают массивы и их элементы, например `$myarray[] = $x` или `$x = $myarray[$y]`
Знаки равенства и "больше" (`=>`)	Обозначают связанные пары "ключ-значение" в ассоциативном массиве, например `$a = array('name' => 'Fred');`
Восклицательный знак (`!`)	Обозначает неравенство, например, в выражении `if ($x != $y)`
Двойная вертикальная черта (`||`)	Обозначает логический оператор ИЛИ, например, в выражении `($a == $b || $a == $c)`
Одинарные или двойные кавычки (`'` или `"`)	Заключают строковые значения. Для этого можно использовать любые кавычки, хотя поведение значений, заключенных в двойные кавычки, оказывается иным, чем в одинарных кавычках
Знак денежной единицы (`$`)	Обозначает начало имен всех переменных, например `$x`
Знак коммерческое "при", а проще говоря, "собачка" (`@`)	Используется в документирующих блоках для обозначения переменных составления документации, например `@package`, `@param`, `@return`
Запятая (`,`)	Служит для разделения значений в списке аргументов метода и элементов массива
Точка (`.`)	Обозначает оператор сцепления (т.е. соединения) символьных строк
Знак равенства (`=`)	Обозначает оператор присваивания значения переменной, например, в выражении `$a = $b;` переменной `$a` присваивается значение переменной `$b`
Двойной (`==`) или тройной (`===`) знак равенства	Обозначает оператор *равенства*. Двойной знак равенства обозначает, что сравниваемые значения равнозначны, а тройной знак равенства, — что они не только *равны*, но и *однотипны*
Знаки "больше" и "меньше" (`>`, `<`, `>=`, `<=`)	Обозначают, что одно из сравниваемых значений больше, меньше, больше или равно, меньше или равно другому значению

Окончание табл. А.1

Символ	Назначение
Вопросительный знак (?) и двоеточие (:)	Используются совместно в тернарном операторе, например `$a = ($b == $c) ? $d : $e;`
Обратная косая черта (\)	Служит для экранирования одних кавычек в других, например `$a = 'It\'s cool';`
Знаки "плюс", "минус", звездочка и косая черта (+, –, *, /)	Обозначают арифметические операторы сложения, вычитания, умножения и деления соответственно
Одинарный амперсанд (&)	Обозначает передачу значения по ссылке в сигнатуре метода
Двойной амперсанд (&&)	Обозначает логический оператор И, например, в выражении `if ($a == $b && $c == $d)`

Наиболее употребительные операторы

В этом разделе вкратце описываются наиболее употребительные операторы языка PHP.

Арифметические и строковые операторы

В языке PHP операторы **+**, **–** , *****, **/** выполняют арифметические операции сложения, вычитания, умножения и деления соответственно. А оператор точки (**.**) служит для сцепления (соединения вместе) двух символьных строк. Ниже приведены характерные примеры применения этих операторов в коде PHP.

```
$a = 'dog';
$b = 'cat';
$c = $a . $b; // переменной $c присваивается строка 'dogcat'
```

Оператор **++** выполняет приращение значения на единицу (`$a++;`), оператор **+=** — приращение одного значения на другое значение, а оператор **.=** — сцепление строковых значений. Ниже приведены характерные примеры применения этих операторов в коде PHP.

```
$a += $b; //  то же, что и $a = $a + $b
$c .= $d; // то же, что и $c = $c . $d
```

Задание и приведение типов переменных

Для задания или приведения типов переменных в качестве оператора используются круглые скобки. Ниже приведены характерные примеры применения такого оператора в коде PHP.

```
$int = (int) 'abc'; // переменной $int присваивается нулевое значение
$int = (int) '123abc'; // переменной $int присваивается целое значение 123
```

Если первая группа символов в строке может быть интерпретирована как целочисленное значение, то именно это значение используется в коде. В противном случае выбирается нулевое значение.

Логические операторы

Не следует забывать, что одинарный знак равенства служит для присваивания переменной заданного значения, тогда как двойной и тройной знаки равенства — для логического сравнения на равенство значений. Ниже приведены характерные примеры применения этих знаков в коде PHP.

```
$a = $b; // присваивает переменной $a значение переменной $b
($a == $b) // истинно, если значения переменных $a и $b равнозначны
($a === $b) // истинно, если значения переменных $a и $b равны и однотипны
```

Для сравнения значений на неравенство дополнительно указывается восклицательный знак (**!**), например (`$a != $b`) или (`$a !== $b`). Если же требуется выяснить, какое из сравниваемых значений больше или меньше, используются операторы **<** и **>** соответственно. А операторы **<=** и **=>** позволяют соответственно выяснить, больше или равно либо меньше или равно одно из двух сравниваемых значений.

Логические выражения связываются с помощью оператора **||** или **OR**, выполняющего операцию логического ИЛИ, а также оператора **&&** или **AND**, выполняющего операцию логического И. Ниже приведены характерные примеры применения этих логических операторов в коде PHP.

```
($a == $b || $c == $d)
($a == $b OR $c == $d)
($a == $b && $c == $d)
($a == $b AND $c == $d)
```

Условные операторы

Для выполнения кода по заданному условию чаще всего применяются условные операторы `if`. Ниже приведен пример простейшей синтаксической формы условного оператора `if`.

```
if ($a == $b)
{
  // выполнить эти операторы, если условие истинно
  $c = d;
  $e = $f;
}
else
{
  // иначе выполнить эти операторы
  $g = $h;
  $i = $j;
}
```

Блок условного оператора `else` является необязательным. Для ввода дополнительных условий в последующих кодовых блоках служит условный оператор `elseif`, как показано в приведенном ниже примере.

```
if ($a == $b)
{
  $x = $y; // если условие истинно
}
```

```
elseif ($c == $d)
{
  $g = $h; // только если не истинно условие ($a == $b) и($c равно $d)
}
```

Блоков условного оператора `elseif` может быть сколько угодно. Следует, однако, предупредить, что для сравнения значений должны использоваться двойные (==), а не одинарные (=) знаки равенства. Последние служат для присваивания значений переменным и возврата присвоенного значения. Рассмотрим в качестве примера следующий фрагмент кода:

```
$a = 'dog';
$b = 'cat';
if ($a = $b) // по ошибке использован знак "=" вместо знака "=="!
{
    echo "true\n"; // результат всегда истинен
    echo $a . "\n"; // теперь переменная $a содержит строку 'cat'
}
```

В этом фрагменте кода вместо одинарного знака равенства по ошибке указан двойной. Оператор, обозначаемый одинарным знаком равенства, выполняет две функции: во-первых, присваивает переменной `$a` значение переменной `$b`; а во-вторых, возвращает новое значение переменной `$a`. Таким образом, выражение `$a = $b` возвращает символьную строку `'cat'`, которая оценивается как логическое значение `true`. Поэтому условие оператора `if` оказывается истинным.

Одинарные знаки равенства можно использовать в условии оператора `if` в том случае, если результат выполнения метода присваивается переменной. Например, при вызове метода `loadResult()` из класса `JDatabase` возвращается конкретное значение или `null`, если результат запроса базы данных окажется пустым. Поэтому значение, возвращаемое методом `loadResult()`, может быть использовано в условии оператора `if` следующим образом:

```
if ($link = $db->loadResult()) {
    // истинно, если по запросу возвращается ненулевой результат
}
```

Рассмотрим этот пример кода более подробно, поскольку он сложнее предыдущих. В этом коде, по существу, выполняются три операции: во-первых, вызывается на выполнение метод `loadResult()` из класса `JDatabase`; во-вторых, возвращаемое значение присваивается переменной `$link`; и в-третьих, это значение служит условием для выполнения оператора `if`. Этот код можно было бы написать иначе, как показано ниже.

```
$link = $db->loadResult();
if ($link) {
    // значение переменой $link истинно, если по запросу
    // возвращается действительный результат
}
```

Следует, однако, предупредить, что если по запросу возвращается нулевое значение как действительный результат, то рассматриваемый здесь код будет действовать совсем не так, как предполагалось. Дело в том, что нулевое значение оценивается в условии оператора `if` как ложное. Но выйти из этого затруднения позволяет приведенный ниже код.

```
$link = $db- >loadResult();
if ($link !== null) {
    // ложно лишь в том случае, если по запросу
    // возвращается пустой результат
}
```

В некоторых случаях вместо простой формы условного оператора `if` удобно пользоваться тернарным оператором языка PHP, как показано в приведенном ниже примере.

```
$a = ($b == $c) ? $d : $e; // если истинно, возвращает $d, а иначе — $e
```

В этой строке кода переменной `$a` присваивается значение переменной `$d`, если условие истинно. В противном случае ей присваивается значение переменной `$e`.

Оператор выбора `switch`

Структура оператора `switch` отлично подходит для выполнения различных кодовых блоков, исходя из значения одного выражения. Ниже приведена основная форма синтаксиса этого оператора.

```
switch ($a)
{
  case 'abc':
    $b = $c; // выполнить операторы, если $a равно 'abc'
    break;
  case 'def':
    $d = $e; // выполнить операторы, если $a равно 'def'
    break;
  default:
    $f = $g; // иначе выполнить этот оператор
    break;
}
```

Операторы `break` нужны для того, чтобы прервать выполнение оператора на выбранной ветви `case`. Так, если опустить оператор `break` в первой ветви `case`, когда `$a` равно `'abc'`, то будет также выполнено присваивание `$d = $e;` во второй ветви.

Операторы цикла

В этом разделе вкратце описываются операторы цикла, применяемые в языке PHP.

Циклы `foreach`

Самый простой способ организовать циклическое обращение к элементам массива — воспользоваться оператором цикла `foreach`. Кодовый блок в цикле `foreach` выполняется по одному разу для каждого элемента массива. Ниже приведен пример циклического обращения к элементам простого цикла `$groups`.

```
foreach ($groups as $group) // переменная $groups должна быть массивом!
{
  echo $group; // переменная $group содержит n-й элемент массива
  // выполнить дополнительные действия, если потребуется
}
```

Если в данном примере переменная $groups не является массивом, то от интерпретатора языка PHP будет получено соответствующее предупреждение. Если же массив пустой, то цикл будет пропущен.

Циклы foreach можно использовать и для обращения к элементам ассоциативного массива. С этой целью достаточно указать переменные, используемые для каждой пары "ключ-значение", как в приведенном ниже примере.

```
$assoc = array('name' => 'Fred', 'age' => 29);
foreach ($assoc as $key => $value) {
  echo $key . ': ' . $value . "\n"; // выводит каждую пару "ключ-значение"
}
```

В данном примере ключ к каждому элементу ассоциативного массива хранится в переменной $key, а значение каждого элемента массива — в переменной $value.

Циклы for

Оператор for позволяет организовать циклы, выполняющиеся *n* раз. Ниже приведен характерный пример такого цикла.

```
// Этот цикл начинается с 0, на каждом его шаге значение
// переменной $i увеличивается на 1, и так продолжается до тех пор,
// пока условие цикла перестанет быть истинным
for( $i = 0; $i < $len; $i++)
{
  echo $i; // этот оператор будет выполняться до тех пор,
           // пока значения переменных $i и $len не окажутся равными
}
```

В данном примере переменная $i служит в качестве счетчика шагов цикла, а ее приращение осуществляется с помощью оператора инкремента $i++. Цикл выполняется до тех пор, пока выражение $i < $len истинно.

Циклы do/while

В циклах do/while кодовый блок допускается выполнять до тех пор, пока условие не перестанет быть истинным. В одной разновидности этих циклов сначала выполняется кодовый блок, а затем проверяется условие, чтобы выяснить, следует ли повторить цикл. Ниже приведен пример организации цикла do-while.

```
do
{
  // Кодовый блок выполняется один раз, а затем повторяется,
  // если условие while данного цикла истинно.
  // Выполнение кодового блока должно в конечном итоге привести
  // к тому, чтобы условие while данного цикла стало ложным!
}
while ($a != $b);
```

В другой разновидности циклов do/while сначала проверяется условие данного цикла, а затем выполняется кодовый блок. Ниже приведен пример организации цикла while.

```
while ($a != $b)
{
```

```
  // Кодовый блок будет выполняться только в том случае,
  // если условие данного цикла истинно.
  // Выполнение кодового блока должно в конечном итоге привести
  // к тому, чтобы условие данного цикла стало ложным!
}
```

При организации обоих разновидностей циклов `do/while` очень важно иметь в виду, что выполнение кодового блока *должно* в конечном итоге привести к тому, чтобы условие данного цикла стало *ложным*, иначе цикл окажется бесконечным.

Оператор `continue`

Этот оператор служит для пропуска остальной части кодового блока, если на одном из шагов цикла выполняется заданное условие. Ниже приведен характерный пример применения оператора `continue` в цикле.

```
foreach ($groups in $group)
{
  if ($group = 'Public') {
    continue; // пропустить вывод группы 'Public'
  }
  echo $group;
}
```

Альтернативный синтаксис

В файлах, содержащих только исходный код PHP, весь этот код обычно размещается в одном элементе разметки. Но в файлах компоновки элементы разметки в коде PHP и HTML используются совместно, и поэтому в таких файлах может присутствовать немало элементов разметки в коде PHP. В зависимости от конкретной ситуации может возникнуть потребность выводить элементы разметки в коде HTML из кодовых блоков условных операторов `if` и циклов.

В подобных случаях обычно рекомендуется использовать альтернативный синтаксис условных операторов `if` и циклов. Это дает возможность выводить элементы разметки в коде HTML из элементов разметки в коде PHP, не заключая первые в кавычки. Благодаря этому исключается также появление дескриптора PHP, содержащего только открывающую или закрывающую фигурную скобку, как, например, `<?php }>`. Такую скобку легко упустить из виду при просмотре и редактировании исходного кода. В табл. А.2 приведены для сравнения примеры стандартного и альтернативного синтаксиса языка PHP. Следует, однако, иметь в виду, что в альтернативном синтаксисе комментарии должны размещаться внутри элементов разметки в коде PHP.

Очень важную роль играют пробелы за пределами элементов разметки в коде PHP, хотя внутри них указывать пробелы совсем не обязательно. Например, две приведенные ниже строки кода дадут совершенно разные результаты.

```
<p>This is a <?php echo 'test'; ?></p>
<p>This is a<?php echo 'test'; ?></p>
```

В первой строке кода между буквой **a** и угловой скобкой **<** есть пробел, а во второй строке — отсутствует.

Таблица А.2. Примеры применения альтернативного синтаксиса языка PHP

Стандартный синтаксис	Альтернативный синтаксис

`// стандартный синтаксис`
`if ($a == $b)`
`{`
`  echo 'equal';`
`} else {`
`  echo 'not equal';`
`}`
`// стандартный синтаксис цикла foreach`
`foreach ($groups as $group)`
`{`
`  $a = $b;`
`}`
	`<?php // альтернативный синтаксис ?>`
	`<?php if ($a == $b) : ?>`
	`  <p><?php echo 'equal'; ?></p>`
	`<?php else : ?>`
	`  <p><?php echo 'not equal'; ?></p>`
	`<?php endif; ?>`
	`<?php // альтернативный синтаксис цикла foreach ?>`
	`<?php foreach ($groups as $group) : ?>`
	`<p><?php echo $group; ?></p>`
	`<?php endforeach; ?>`

Переменные

Имена всех переменных в языке PHP начинаются со знака денежной единицы (**$**), после которого следует буква или знак подчеркивания (например, `$myvariable` или `$_myvariable`), а после них можно указывать цифры и тире. В именах переменных учитывается регистр, а следовательно, `$myvariable` и `$myVariable` — это разные переменные. Поэтому следует быть очень внимательным, чтобы не ошибиться, обозначая переменную в другом регистре. И это одна из причин, по которым именовать переменные следует в определенном порядке, пользуясь, например, смешанным написанием вроде `$myVariableName`.

Объявление переменных

Объявлять переменные в языке PHP совсем не обязательно, но следует непременно присвоить им значения, прежде чем использовать их в выражении. В противном случае интерпретатор языка PHP выдаст предупреждение.

Область действия переменных

Как правило, переменные определяются в области действия того кодового блока, где они устанавливаются. Так, если переменная устанавливается в самом внешнем кодовом блоке сценария, то она окажется в области действия всех остальных кодовых блоков этого сценария, включая и кодовые блоки условных операторов `if` и циклов. Если же исходный файл включается в код по команде `include` или `require`, то переменные остаются в области действия включаемого файла. Именно поэтому переменные из представления по-прежнему остаются в области его действия при вызове метода `loadTemplate()` для загрузки файлов компоновки. А если переменные определены в методе или функции класса как локальные, то они остаются в области действия только этого метода или функции.

Массивы

Массивы могут быть созданы как с начальными значениями своих элементов, так и без таковых. В приведенной ниже строке кода создается пустой массив.

```
$emptyArray = array();
```

А в следующей строке кода создается массив с двумя элементами, которым присваиваются определенные значения:

```
$simple = array('zero', 'one');// $simple[0] = 'zero', $simple[1] = 'one'
```

Массивы могут быть индексированными или ассоциативными. В индексированных массивах целые числа (**0**, **1**, **2** и т.д.) служат для доступа к их элементам. Например, в приведенном ниже коде осуществляется доступ к первому элементу массива, созданного в предыдущем примере. Не следует, однако, забывать, что первый элемент массива указывается по индексу **0**, а не **1**.

```
echo $simple[0]; // вывести символьную стоку 'zero'
```

В ассоциативных массивах хранятся пары "ключ–значение". В приведенной ниже строке кода создается ассоциативный массив.

```
$assoc = array('name' => 'Jim', 'age' => 29);
```

Доступ к элементам ассоциативного массива осуществляется следующим образом:

```
echo $assoc['name']; // выводит имя Jim
echo $assoc['age']; // выводит значение 29
```

Для ввода нового элемента в индексированный массив служат квадратные скобки (**[]**), как показано ниже.

```
$simple[] = 'new value'; // добавить в конец массива
```

А для ввода нового элемента в ассоциативный массив служит вновь добавляемый ключ:

```
$assoc['telephone'] = '123-45-67';
```

Для удаления элемента из массива служит команда `unset` и соответствующая ей функция, как показано в приведенном ниже примере.

```
unset($assoc['age']) // удаляет элемент 'age' из ассоциативного массива
```

Элементы массива могут разнотипными, в том числе объектами и другими массивами. Например, в приведенном ниже фрагменте кода создается индексированный массив, элементами которого являются ассоциативные массивы.

```
$person1 = array('name' => 'Jim', age => 29);
$person2 = array('name' => 'Jill', age => 27);
$people = array($person1, $person2);
```

В данном примере доступ к элементу внутреннего массива осуществляется следующим образом:

```
echo $people[0]['name']; // выводит имя Jim
```

Здесь первый элемент `$people[0]` внешнего массива является первым ассоциативным массивом. Следовательно, при обращении `$people[0]['name']` к первому элементу внутреннего массива на экран выводится хранящееся в нем имя `Jim`.

Обращение с массивами

Ниже приведен ряд примеров обращения с массивами, для чего используются функции `count()`, `is_array()` и `isset()`.

```
$count = count(array(1,2,3)); // количество элементов массива равно трем
is_array($a) // возвратить логическое значение true, если $a — это массив,
             // а иначе — логическое значение false
isset($person1['name']) // возвратить логическое значение true, если элемент
                        // существует, а иначе — логическое значение false
```

Дополнительные сведения о функциях обращения с массивами можно найти по адресу `http://php.net/manual/en/ref.array.php` или `http://php.net/manual/ru/ref.array.php`.

Символьные строки

Символьные строки представляют собой переменные, содержащие символы, в том числе буквы, числа и знаки. В языке PHP имеется целый ряд встроенных методов для обработки символьных строк, хотя не все из них корректно обращаются с наборами символов в коде UTF-8. В классе `JString`, объявляемом в файле `libraries/joomla/utilities/string.php`, содержатся аналогичные методы, специально предназначенные для правильного обращения с символами в коде UTF-8. Именно поэтому для обработки строк, которые могут содержать символы в коде UTF-8, следует пользоваться методами из класса `JString`. Ниже приведены некоторые примеры обработки символьных строк.

```
$pos = JString::strpos('abcd', 'bc'); // результат должен быть равен 1
$len = JString::strlen('abc'); // результат должен быть равен 3
$replaced = str_replace('dog', 'cat', 'I love dogs.'); // Я люблю кошек.
```

Дополнительные сведения о функциях обработки символьных строк в языке PHP можно найти по адресу `http://php.net/manual/en/ref.strings.php` или `http://php.net/manual/ru/ref.strings.php`, а о классе `JString` — по адресу http://api.joomla.org/Joomla-Platform/String/JString.html.

Константы и текущий каталог

Константы могут быть определены в языке PHP с помощью оператора `define`, как показано в приведенном ниже примере.

```
define('_JEXEC', 1);
```

В данном примере определяется константа `_JEXEC` со значением **1**. Напомним, что эта константа используется практически во всех файлах Joomla для выяснения того факта, что код выполняется в приложении Joomla. Однажды определенная константа не может быть изменена. Имена определяемых констант принято указывать только прописными буквами.

В языке PHP предопределен целый ряд специальных, так называемых "волшебных" констант. К их числу относится константа __FILE__, задающая полный путь к текущему файлу. В языке PHP имеется также функция dirname(), возвращающая имя каталога для любого файла, а по существу, — часть полного пути к файлу до последней косой черты, исключая его имя. Константу __FILE__ и функцию dirname() можно использовать совместно для получения каталога, в котором находится текущий файл. Например, в приведенной ниже строке кода определяется константа JPATH_BASE с именем каталога, в котором находится текущий файл.

```
define('JPATH_BASE', dirname(__FILE__));
```

Кроме того, в языке PHP имеется функция file_exists(), проверяющая наличие или отсутствие искомого файла в файловой системе. Дополнительные сведения о функциях PHP для работы с файловой системой можно найти по адресу http://php.net/manual/en/ref.filesystem.php или http://php.net/manual/ru/ref.filesystem.php.

Функции и методы

Большая часть кода программ, написанных для Joomla, содержится в теле функций, которые, в свою очередь, находятся в классах. Но иногда функции создаются как автономные единицы программного кода и не входят в состав классов. Функции, объявляемые в классах, зачастую называются методами (именно так их принято называть в языках ООП), хотя многие пользуются обозначениями "функция" и "метод" попеременно. Следует также заметить, что в языке PHP зарезервировано ключевое слово только для функции, но не для метода.

Структура функции

У функции имеется следующая структура:

```
function <имя функции> (<аргументы функции, если таковые имеются>)
{ // начало тела функции
    <код функции, если таковой имеется>
} // конец тела функции
```

Ниже приведен пример определения простой функции.

```
public function setLastVisit($timestamp = null)
{
  // создать объект таблицы пользователей
  $table = $this->getTable();
  $table->load($this->id);
  return $table->setLastVisit($timestamp);
}
```

В определении этой функции указан модификатор доступа public, а это означает, что ее можно вызывать из любого класса. У данной функции имеется единственный аргумент $timestamp, который по умолчанию имеет пустое значение null. Следовательно, данную функцию можно вызывать, по существу, без аргументов. Но если не указать значение по умолчанию при вызове функции setLastVisit(), то будет получено сообщение об ошибке. Тело данной функции состоит из трех строк кода. В последней строке кода

используется оператор `return` для возврата значения. Это означает, что данную функцию можно вызвать в операторе присваивания, как показано ниже.

```
$x = $user->setLastVisit($myTimestamp);
```

В данном случае значение, возвращаемое функцией `setLastVisit()`, сохраняется в переменной `$x`. Количество и порядок следования аргументов в вызове функции или метода называется *сигнатурой*.

Область действия переменной в функции

Имена переменных в теле функции связаны только с самой функцией. В предыдущем примере при вызове функции `setLastVisit()` в качестве аргумента ей передавалась переменная `$myTimestamp`. В теле данной функции переменная `$myTimestamp` не определена, но ее значение известно по имени переменной `$timestamp`. По завершении данной функции переменная `$timestamp` оказывается неопределенной. По желанию в теле функции можно было бы воспользоваться тем же самым именем переменной, что и при вызове функции. Но и в этом случае обе переменные были бы разными, поскольку у них разные области действия, хотя и одинаковые имена. Одна из них определяется в теле функции, а другая — за пределами функции.

Передача значений по ссылке

Предупреждение

Этот способ передачи значений чреват недоразумениями. Когда переменная передается функции или методу в качестве аргумента, то, как правило, она оказывается локальной в теле этой функции или метода. Это означает, что если значение данной переменной изменяется в теле функции или метода, то внесенные изменения теряются при возврате из функции или метода.

Если переменная передается функции или методу по ссылке с помощью префикса **&**, то при изменении ее значения в теле функции или метода внесенные изменения становятся доступными в коде, из которого вызывалась функция или метод. Обратимся к следующему примеру:

```
function pluralValue($string) {
  $string .= 's';
}
function pluralReference(&$string) {
  $string .= 's';
}
$a = 'cat';
pluralValue($a);
echo $a . "\n"; // переменная $a по-прежнему содержит строку "cat"
pluralReference($a);
echo $a . "\n"; // а теперь переменная $a содержит строку "cats"
```

При возврате из функции `pluralValue()` изменения локальной переменной теряются. В то же время изменения локальной переменной в функции `pluralReference()` становятся доступными в вызывающем коде, поскольку в сигнатуре этой функции указан префикс **&**, обозначающий передачу ей значения по ссылке.

Включение файлов и их типы

В такой крупной кодовой базе, как Joomla, исходный код приходится распределять по многим файлам. А для этого нужно разбираться в различных типах файлов PHP, знать, как они включаются в работающую программу и что при этом происходит. Практически все файлы в Joomla разделяются на три категории.

- Объявление класса.
- Объявление функции.
- Простой сценарий.

Файлы объявления классов

В ООП для каждого класса, как правило, выделяется отдельный файл. Подавляющее большинство файлов в системе управления информационным наполнением (CMS) вебсайтов и на платформе Joomla организованы именно таким способом. В каждом из них объявляется отдельный класс. Ниже приведена структура объявления класса.

```
class <имя класса>
{ // начало кодового блока класса
  <поля класса, если таковые имеются>
  <методы класса, если таковые имеются>
} // конец кодового блока класса
```

Как правило, в начале каждого файла, где объявляется класс, присутствует одна из следующих строк кода:

```
defined('JPATH_PLATFORM') or die; // используется в файлах платформы
defined('_JEXEC') or die; // используется в файлах системы CMS
```

В этих строках кода гарантируется, что данный файл выполняется в среде Joomla. Зачастую после такой строки кода следует один или более оператор `import` для включения файлов классов, требующихся для данного класса. Когда файлы объявления классов включаются с помощью оператора `include`, `require`, `jimport` или функции `JLoader::register()`, соответствующие классы и их методы вводятся в рабочую область памяти, но программный код не выполняется. Единственное изменение состоит в том, что класс вместе со своими методами и полями становится доступным для применения в той программе, в которую он включен.

Файлы объявления функций

В некоторых файлах, используемых в Joomla, вместо классов объявляются функции. К их числу относится файл `templates/system/html/modules.php`, в котором предоставляются основные функции для стилевой рамки модуля, а также файл `components/com_content/router.php`, в котором предоставляются функции `ContentBuildRoute()`

и `ContentParseRoute()`. Эти файлы могут быть включены только по командам `include` или `require`, а также их разновидностям `include_once` и `require_once`. А поскольку в них не объявляются классы, то они и не могут быть включены с помощью методов `jimport()` или `JLoader::register()`.

При включении файлов объявления функций, как, впрочем, и файлов объявления классов, код вообще не выполняется. Вместо этого объявленные в них функции вводятся в рабочую область памяти, где они становятся доступными для вызова из вызывающего сценария.

Простые сценарии

Файлы простых сценариев содержат последовательный ряд операторов языка PHP. К их числу относятся файлы `index.php` для пользовательской и административной частей веб-сайта на Joomla, а также файлы шаблонов и компоновки. Как и файлы объявления функций, файлы простых сценариев не содержат никаких объявлений классов, а следовательно, они могут быть включены по команде `include` или `require`, а также их разновидностям `include_once` и `require_once`. Но в отличие от других типов файлов, когда простые сценарии включаются в программу, их операторы выполняются немедленно. А любые переменные, находящиеся в области действия той части программы, в которую включается простой сценарий, по-прежнему остаются в области действия сценария.

Смешанные файлы

В одном файле можно также сочетать операторы простого сценария с объявлениями функции или класса. В этом случае объявленный класс или функция вводится в рабочую область памяти, а операторы, находящиеся за пределами кодовых блоков, выполняются немедленно. Такие файлы должны быть включены по команде `include` или `require`, а также их разновидностям `include_once` и `require_once`.

Включение файлов

Для включения других файлов в прикладную программу Joomla, выполняющуюся в данный момент, служат следующие команды и функции.

- Команда `include`: включает файл любого типа, но вызывает ошибку, если во включаемом файле объявляется класс или функция, которые уже объявлены в другом месте. Если же найти включаемый файл не удастся, это не приведет к ошибке.
- Команда `include_once`: действует так же, как и команда `include`, за исключением того, что в ней проверяется, был ли данный файл включен раньше. На некоторых серверах эта команда может выполняться медленнее, поэтому для включения файлов объявления классов лучше пользоваться командой `jimport` или `JLoader::register()`.
- Команда `require`: действует так же, как и команда `include`, за исключением того, что она приводит к ошибке, если файл не удается найти.
- Команда `require_once`: действует так же, как и команда `require`, за исключением того, что она не пытается включить файл, который был включен раньше. И эта команда может выполняться медленнее на некоторых серверах, поэтому лучше пользоваться командой `jimport()` или `JLoader::register()`.

- Команда `JLoader::register()`: обеспечивает более предпочтительный способ загрузки файлов объявления классов, но для этого требуется знать имя класса и файла, в котором он объявляется. Используется в механизме автозагрузки языка PHP для повышения производительности.
- Команда `jimport`: обеспечивает простой способ вызова функции `JLoader::register()`, но требует, чтобы имена файла и класса соответствовали принятым в Joomla условным обозначениям. Это наиболее распространенный способ включения файлов на платформе Joomla.

Основы ООП

Среда Joomla разработана по принципам объектно-ориентированного программирования (ООП) и соответствующим шаблонам проектирования. Поэтому ниже дается краткое введение в ООП.

Классы и объекты

Классы являются стандартными блоками объектно-ориентированных программ. Класс можно рассматривать в качестве шаблона для создания объектов данного типа. Так, класс `JDocumentHTML` служит для создания переменной (например, `$doc`) ссылки на объект типа `JDocumentHTML`. Как правило, классы содержат поля и методы, иначе называемые функциями. Для вызова метода служит следующий синтаксис:

```
$x = $doc->getHead();
```

где `getHead()` — имя метода из текущего класса. А для доступа к полю класса служит такой синтаксис:

```
$x = $doc->template;
```

где `$template` — поле из текущего класса. Обратите внимание на то, что для доступа к полю знак `$` опускается.

Метод-конструктор

В большинстве классов определяется специальный метод под названием `__construct()`. Этот метод вызывается автоматически при создании объекта данного типа. Он содержит код, требующийся для инициализации объекта, например, для установки значений по умолчанию в полях объекта.

Создание объектов

Прежде чем воспользоваться объектом, необходимо создать его. Большинство объектов создаются в Joomla тремя способами.

- Вызов метода `JFactory::get<имя>()`. Класс `JFactory` служит для создания наиболее часто используемых глобальных объектов таких классов, как `JApplication`, `JConfig`, `JDate`, `JSession`, `JLanguage`, `JDocument`, `JUser`, `JCache`, `JAcl`, `JDbo`, `JMailer`, `JFeedParser`, `JXMLParser`, `JXML`, `JEditor` и `JURI`. Например, в приведенной ниже строке кода получается текущий объект типа `JApplication`.
  ```
  $app = JFactory::getApplication();
  ```

- Вызов метода `getInstance()`. Во многих классах Joomla предоставляется метод `getInstance()`. В этом методе, как правило, проверяется, существует ли объект данного типа. И если такой объект существует, то данный метод возвращает уже созданный объект, а иначе — новый объект. Ниже приведен пример создания объекта данным способом.
  ```
  $client = JTable::getInstance('Client','BannersTable');
  ```
- Применение оператора `new`. Этот оператор служит для создания объектов иным способом, чем с помощью класса `JFactory` и метода `getInstance()`. В данном случае новый экземпляр объекта создается вызовом конструктора его класса. У конструкторов некоторых классов имеются как обязательные, так и необязательные аргументы. Ниже приведен пример создания объекта с помощью оператора `new`.
  ```
  $registry = new JRegistry();
  ```
- Указывать круглые скобки необязательно, если аргументы не передаются конструктору.

Стандартный класс

В языке PHP имеется встроенный класс, называемый *стандартным*. Объект этого класса можно создать следующим образом:

```
$person = new stdClass(); // как с круглыми скобками, так и без них
```

В стандартном классе определены встроенные методы для создания полей. Например, в приведенном ниже фрагменте кода создаются поля `name` и `age` и в них устанавливаются значения.

```
$person->name = 'Bob';
$person->age = 15;
```

Аналогичным образом значения извлекаются из полей объекта, как показано ниже.

```
$age = $person->age;
```

Стандартные классы удобны для хранения списка свойств объекта.

Расширение классов и наследование

Наследование является основополагающим принципом ООП. Оно позволяет создать новый класс в качестве подкласса, производного от другого (родительского) класса, используя ключевое слово `extends`. Обратимся к следующему примеру:

```
class JDocumentHTML extends JDocument
```

В данном примере создается подкласс `JDocumentHTML`, являющийся производным от класса `JDocument`. Таким образом, класс `JDocumentHTML` автоматически наследует *все* поля из своего родительского класса `JDocument`. Так, если требуется создать подкласс, аналогичный родительскому классу, но определить в нем только один новый метод, то достаточно лишь написать код этого метода, а все остальные методы будут доступны из родительского класса. Это все равно, что скопировать и вставить все методы из родительского класса в производный от него класс, но при этом не придется сопровождать обе копии кода.

Например, подкласс JSite является производным от класса JApplication, а следовательно, он наследует все методы из своего родительского класса. Так, в классе JApplication имеется метод getInstance(). Если проанализировать исходный код класса JSite, объявляемого в файле includes/application.php, то в нем не обнаружится метод getInstance(). Тем не менее этот метод можно вызывать для объекта класса JSite. В этом случае интерпретатор языка PHP найдет метод getInstance() в родительском классе.

В Joomla имеется немало примеров наследования за счет расширения одних классов другими. Кроме того, одни подклассы могут быть производными от других подклассов. Например, класс JApplication является производным от класса JObject. Следовательно, у класса JSite имеется родительский класс JApplication и прародительский класс JObject. Наследование классов может быть произвольной глубины.

Переопределение методов

При создании подкласса в него можно ввести новые методы и поля, а также изменить уже имеющиеся методы и поля. Такой прием называется переопределением. Для переопределения достаточно создать в подклассе метод с тем же самым именем и модификатором доступа (public, protected, private), что и у метода в родительском классе.

Ключевое слово parent имеет специальное назначение. При переопределении метода можно воспользоваться выражением parent::<имя метода> для вызова соответствующего метода из родительского класса. Ниже приведен характерный пример применения ключевого слова parent.

```
parent::loadConfiguration($data);
```

Это может принести пользу при сходстве методов в подклассе и родительском классе. В таком случае достаточно реализовать отличия в обоих методах, а затем вызвать метод из родительского класса.

Модификаторы public, protected и private

Когда в классе создаются поля и методы, они обозначаются модификаторами доступа как открытые (public), защищенные (protected) и закрытые (private).

- Если поля и методы открытые, это означает, что они доступны из любого другого класса или сценария.
- Если поля и методы защищенные, это означает, что они доступны только из своего класса или его подклассов.
- Если поля и методы закрытые, это означает, что они доступны только из своего класса.

Статические методы, поля и переменные

Статические методы вызываются из самого класса, а не его объекта, как в приведенном ниже примере.

```
$now = JFactory::getDate();
```

В данном примере метод `getDate()` является статическим и вызывается с указанием имени его класса через двойное двоеточие (`::`). Статические методы объявляются с ключевым словом `static`, как показано ниже.

```
public static function getDate($time = 'now', $tzOffset = null)
```

Статическим называется такое поле, которое объявляется с ключевым словом `static` и вызывается из класса. Ниже приведен характерный пример объявления статического поля.

```
public static $format = 'Y-m-d H:i:s';
```

Для доступа к статическому полю за пределами его класса служит следующий синтаксис:

```
<имя класса> + :: + <имя переменной, включая знак "$">
```

Ниже приведен характерный пример обращения к статическому полю за пределами его класса.

```
$format = JDate::$format;
```

Переменная `$this` и ключевые слова `self` и `parent`

Специальная переменная `$this` служит для обращения к текущему объекту из его класса. Так, в приведенном ниже примере метод `getModel()` вызывается из того же самого класса.

```
$model = $this->getModel();
```

А в следующей строке кода полю `filter` из того же самого класса присваивается значение, извлекаемое по ключу `filter` из ассоциативного массива.

```
$this->filter = $options['filter'];
```

Ключевое слово `self` служит для вызова статического метода или доступа к статическому полю из текущего класса. В приведенном ниже примере статический метод `getGroupsByUser()` вызывается из того же самого класса (в данном случае — из класса `JAccess`).

```
$identities = self::getGroupsByUser($userId);
```

А в следующей строке кода возвращается значение из статического поля `$_buffer` того же самого класса (в данном случае — класса `JDocument`).

```
return self::$_buffer;
```

Ключевое слово `parent` служит для вызова метода или доступа к полю в родительском классе. Так, в приведенном ниже примере метод `display()` вызывается из родительского класса.

```
parent::display();
```

А в следующей строке кода возвращается значение из поля `$_buffer` того класса, который является родительским для текущего класса.

```
return parent::$_buffer;
```

Следует иметь в виду, что ключевое слово `parent` буквально означает "любой суперкласс", включая родительский, прародительский и т.д. Так, если класс `A` является родительским для класса `B`, а тот, в свою очередь, родительским для класса `C`, то класс `A` оказывается прародительским для класса `C`. Допустим, поле `$myfield` и метод `myMethod()` определены в классе `A`, а не в классе `B` или `C`. Если вызвать метод `parent::myMethod()` из родительского класса, то соответствующий код будет выполнен в классе `A`. А если обратиться к полю `parent::$myfield`, то его значение будет извлечено из класса `A`.

Простая отладка программ

Иногда оказывается полезно заглянуть в работающую программу, чтобы проанализировать значения ее переменных. Как пояснялось в главе 2, для этой цели в частности и для отладки программ, написанных на языке PHP, вообще можно воспользоваться интегрированной средой разработки Eclipse или Netbeans. А для более простой отладки программ в исходные файлы PHP можно попробовать ввести временный тестовый код.

Просмотр определенных переменных

Для просмотра всех переменных и установленных в них значений в области их действия можно, в частности, ввести следующий фрагмент кода в нужном месте программы:

```
var_dump(get_defined_vars());
die;
```

Функция `get_defined_vars()` возвращает массив всех переменных, определенных в данном месте выполнения программы. Эта функция выводит из оперативной памяти одну или более переменную в удобочитаемом формате. А команда `die` останавливает выполнение программы в данном месте для просмотра результатов в окне браузера. В конечном итоге выполнение программы останавливается, а переменные, определенные в области действия на момент выполнения программы, выводятся вместе с их значениями.

Просмотр трассировки стека

Для отладки программы не менее важно выяснить, каким образом ее выполнение достигло интересующей нас стадии. Как упоминалось в главе 2, в программном стеке хранится последовательность вызовов функций, приведшая к данной стадии выполнения программы (например, в файле `index.php` был вызван метод `dispatch()` из класса `JSite`, где, в свою очередь, был вызван метод `renderComponent()` из класса `JComponentHelper`). Содержимое стека можно вывести в любой момент выполнения программы с помощью следующего фрагмента кода:

```
var_dump(debug_backtrace());
die;
```

Трассировка стека выводится в следующей последовательности: текущая программа — из вершины стека, а исходная вызывающая программа (в данном случае файл `index.php`) — из дна стека.

Более совершенные приемы программирования

Приведенных выше основ должно быть достаточно для понимания большей части исходного кода в Joomla. Но в некоторых местах кодовой базы Joomla применяются некоторые более совершенные приемы программирования.

Применение логического оператора || вместо условного оператора if

В некоторых местах кодовой базы Joomla можно встретить фрагменты кода, аналогичные следующему:

```
$lang->load('tpl_'.$template, JPATH_BASE, null, false, false)
  || $lang->load('tpl_'.$template, JPATH_THEMES."/$template", null,
↳false, false)
  || $lang->load('tpl_'.$template, JPATH_BASE,
↳$lang->getDefault(),false, false)
  || $lang->load('tpl_'.$template, JPATH_THEMES."/$template",
↳$lang->getDefault(), false, false);
```

В этом фрагменте кода делается последовательный ряд вызовов методов `$lang->load()`, разделенных логическим оператором `||`. Для понимания этого фрагмента кода важно иметь в виду следующее. Во-первых, если вызов метода `$lang->load()` завершается успешно, то возвращается логическое значение `true`. Во-вторых, логический оператор `||` (ИЛИ) прекращает обработку после возврата первого же логического значения `true`. Это означает, что если успешно завершится выполнение хотя бы одного метода `load()`, то остальная часть кода (после логического оператора `||`) пропускается.

Аналогичного результата можно было бы добиться и с помощью ряда условных операторов `if`. Но в этом случае код получился бы более длинным и сложным.

Связывание методов в цепочку

В языке PHP теперь допускается связывание методов в цепочку. Это означает, что если один метод возвращает объект, то этот объект может быть использован для выполнения другого метода. Ниже приведен простой пример связывания методов в цепочку.

```
if (!JFactory::getUser()->authorise('core.admin')) {
```

В данном примере требуется проверить полномочия пользователя, для чего требуется объект типа `JUser`. И этот объект можно получить, вызвав метод `JFactory::getUser()`, а возвращаемый им результат (т.е. объект типа `JUser`) использовать для вызова метода `authorise()`. То же самое можно было бы сделать, написав следующие две строки кода:

```
$user = JFactory::getUser();
if (!$user->authorise('core.admin')) {
```

Но если переменная `$user` не используется ни в каких других целях, кроме ссылки на объект типа `JUser`, то, связав методы в цепочку, можно сэкономить одну строку кода, используя в нем на одну переменную меньше.

Волшебные методы PHP

В состав языка PHP входит целый ряд так называемых "волшебных" методов, автоматически вызываемых в предопределенных местах программ. Имена всех этих методов начинаются с двойного знака подчеркивания (__). Характерным тому примером служит метод __construct(). Этот метод автоматически вызывается всякий раз, когда с помощью оператора new создается новый объект:

```
$r = new JRegistry();
```

Еще одним характерным примером служит метод __call(). Если этот метод определен в функции, то он выполняется в любой момент, когда вызывается недоступный метод. Недоступным может оказаться такой метод, который не определен в классе (или его родительских классах), или же такой метод, модификатор доступа которого (например, private или protected) не допускает его вызов. Метод __call() используется, в частности, для создания псевдонимов существующих методов. Например, в классе JDatabase имеются два метода, называемых quote() и quoteName(). Но для вызова этих методов можно было бы воспользоваться альтернативными именами q() и qn() или nq() соответственно. И такую возможность можно реализовать в классе JDatabase с помощью метода __call(), как показано ниже.

```
public function __call($method, $args)
{
  if (empty($args)) {
    return;
}

   switch ($method)
{
  case 'q':
    return $this->quote($args[0], isset($args[1]) ? $args[1] : true);
    break;
  case 'nq':
  case 'qn':
    return $this->quoteName($args[0]);
    break;
}
}
```

Этот метод вызывается всякий раз, когда предпринимается попытка вызвать метод, несуществующий в данном классе. Имя этого метода указывается в качестве первого аргумента ($method) метода __call(), а остальные его аргументы указываются в массиве $args.

Сначала в данном методе проверяется наличие аргументов. У обоих методов, quote() и quoteName(), должны быть непременно аргументы, а следовательно, их вызов без аргументов недопустим. И если массив $args оказывается пустым, то происходит немедленный возврат из метода, и ничего другого в нем больше не делается. Затем выполняется оператор switch для выбора вызываемого метода по имени. Так, если указано имя 'q', то вызывается метод quote(), которому передается первый аргумент, а также логическое значение true, если имеется и второй аргумент. А если указано имя 'qn' или "nq', то вызывается метод quoteName(), которому передается первый аргумент.

Таким образом, вместо имени метода `quote()` можно воспользоваться псевдонимом `q()`, а вместо имени метода `quoteName()` — псевдонимом `qn()` или `nq()`. Например, метод `q()` можно определить следующим образом:

```
public function q($text, $escape = true) {
  return $this->quote($text, $escape);
}
```

Переменные в качестве имен классов и методов

Еще один более совершенный прием программирования для Joomla состоит в применении переменных вместо имен классов и методов. Например, в методе `execute()` из класса `JController` имеется следующая строка кода:

```
$retval = $this->$doTask();
```

Имя выполняемого метода находится в переменной `$doTask`. Так, если в переменной `$doTask` находится значение `"display"`, то вызывается метод `display()`. Преимущество такого приема заключается, в частности, в том, что не нужно заранее знать имена всех методов, которые могут быть вызваны. При вызове можно воспользоваться именем любого метода, лишь бы он существовал.

Регулярные выражения

Регулярные выражения применяются в операциях расширенного поиска и замены символьных строк. В Joomla они используются в самых разных целях, например, для фильтрации специальных символов, поиска и замены дескрипторов `jdoc:include` в объектах документов, а также видоизменения внутренних ссылок при использовании удобных для поиска URL. Чаще всего регулярные выражения применяются в двух функциях: `preg_match()` и `preg_replace()`. Нетрудно догадаться, что в функции `preg_match()` осуществляется поиск и сопоставление символьных строк с шаблоном, а в функции `preg_replace()` — поиск и замена символьных строк. Ниже приведен характерный пример применения регулярного выражения в функции `preg_match()`.

```
$result = preg_match('#\.(?:bmp|gif|jpe?g|png)$#i', $url);
```

В данном примере проверяется, оканчивается ли переменная `$url` точкой и расширением файла `bmp`, `gif`, `jpe`, `jpeg` или `png`, и в этом случае в переменной `$result` сохраняется логическое значение `true`. Для того чтобы сделать то же самое, пришлось бы иначе воспользоваться целым рядом условных операторов `if`, но для этого потребовалось бы намного больше кода.

Ниже приведен характерный пример применения регулярного выражения в функции `preg_match()`.

```
$result = (string) preg_replace('/[^A- Z_]/i', '', $source);
```

В данном примере отфильтровываются небуквенные символы в символьной строке из переменной `$source`. Если в ней обнаружатся любые символы, кроме букв или знаков подчеркивания, они будут заменены пробелами, а по существу, удалены из данной строки. И этого можно было бы добиться с помощью других функций, но затратив больше кода.

Применение регулярных выражений — довольно сложная и обширная тема, выходящая за рамки рассмотрения этой книги. Правда, в Интернете имеется немало полезных ресурсов для самостоятельного изучения регулярных выражений.

Приложение Б

Типы фильтров в Joomla!

Фильтрация данных относится к числу самых важных аспектов создания безопасных веб-сайтов. Данные, поступающие из таких ненадежных источников, как URL или форма, должны всегда просеиваться с помощью самых строгих и ограничительных фильтров, которые только возможны в данной ситуации.

Встроенные в Joomla! функциональные возможности фильтрации предоставляются в классе `JFilterInput`, объявляемом в файле `libraries/joomla/filter/filterinput.php`. В табл. Б.1 приведены доступные в нем типы фильтров.

Таблица Б.1. Типы фильтров в Joomla

Тип фильтра	Описание
`ALNUM`	Приводит данные к строковому типу и удаляет из них все символы, кроме `A-Z`, `a-z` и `0-9`
`ARRAY`	Приводит данные к типу массива, но не фильтрует элементы массива автоматически
`BASE64`	Делает то же, что и фильтр `ALNUM`, кроме фильтрации знаков "плюс" (+), косой черты (/) и равенства (=)
`BOOL/BOOLEAN`	Приводит значение к логическому типу, чтобы возвратить логическое значение `true` или `false`
`CMD`	Делает то же, что и фильтр `ALNUM`, кроме фильтрации знаков тире (-), точки (.) и подчеркивания (_)
`FLOAT/DOUBLE`	Приводит данные к числовому типу с плавающей точкой. Если во входной символьной строке имеется несколько значений с плавающей точкой, то выбирается только первое из них
`HTML`	Фильтрует элементы разметки в коде HTML с учетом нужного метода фильтрации дескрипторов (по черному или белому списку)
`INT/INTEGER`	Приводит данные к целочисленному типу со знаком

Окончание табл. Б.1

Тип фильтра	Описание
PATH	Отфильтровывает достоверный полный путь к файлу
STRING	Делает то же, что и фильтр HTML, за исключением того, что сначала декодирует любые закодированные значения во входной строке (например, символы, закодированные с помощью функции htmlspecialchars() или htmlentities() языка PHP)
UINT	Приводит данные к целочисленному типу и берет абсолютное значение результата для получения положительного целого числа
USERNAME	Приводит данные к строковому типу и отфильтровывает достоверное имя пользователя, не допуская символы &, <, >, %, '
WORD	Приводит данные к строковому типу, допуская только буквы и знак подчеркивания (_)

Фильтрация элементов разметки в коде HTML

Фильтры типа HTML и **STRING** допускают фильтрацию элементов разметки в коде HTML как по черному, так и по белому списку. Фильтрация по белому списку позволяет определить список *допустимых* элементов разметки в коде HTML и их атрибутов, чтобы удалять любые элементы или атрибуты, которые отсутствуют в данном списке. А фильтрация по черному списку позволяет определить список *недопустимых* элементов и атрибутов, чтобы удалять их, оставляя только те элементы и атрибуты, которые отсутствуют в данном списке.

По умолчанию в Joomla выбирается фильтрация по черному списку. Недопустимыми по умолчанию считаются следующие элементы разметки:

```
applet, body, bgsound, base, basefone, embed, brame, frameset, head,
html, id, iframe, ilayer, layer, link, meta, name, object, script,
style, title, xml
```

В черном списке по умолчанию присутствуют также следующие атрибуты:

```
action, background, codebase, dynsrc, lowsrc
```

Фильтрация данных в приложениях Joomla

Доступ к средствам фильтрации в Joomla осуществляется несколькими способами. В частности, средства фильтрации встроены в классы JForm, JRequest и JInput. А классом JFilterInput можно пользоваться для фильтрации данных непосредственно.

Фильтрация в классе JForm

Используя атрибут filter, можно ввести фильтрацию в элемент разметки при создании XML-файла для компоновки формы типа JForm. Например, приведенный ниже атрибут filter обеспечивает фильтрацию значения, введенного в поле, с помощью фильтра INTEGER.

```
filter="integer"
```

Кроме типов фильтров, приведенных в табл. Б.1, в классе JForm предоставляются типы фильтров, перечисленные табл. Б.2. Помимо этих встроенных типов фильтров, можно также ввести специальные средства фильтрации данных в полях формы типа JForm.

Таблица Б.2. Дополнительные типы фильтров из класса JForm

Тип фильтра	Описание
INT_ARRAY	Фильтрует каждый элемент массива, приводя его к целочисленному типу
RAW	Не выполняет фильтрацию
SAFEHTML	Делает то же, что и фильтр STRING из класса JFilterInput
SERVER_UTC	Преобразует входные данные в метку даты и времени, используя формат базы данных MySQL, в часовой пояс, установленный на сервере
TEL	Отфильтровывает номер телефона
UNSET	Устанавливает исходное (обычно пустое) значение и служит для фильтрации скрытых полей в тех формах, где ввод данных не допускается
USER_UTC	Преобразует входные данные в метку даты и времени, используя формат базы данных MySQL, в часовой пояс, установленный пользователем

Фильтрация в классах JRequest и JInput

В классах JRequest и JInput допускается фильтрация значений с помощью типов фильтров из класса JFilterInput, приведенных в табл. Б.1. Так, в классе JRequest фильтрация организуется в двух формах. В общей форме вызывается метод JRequest::getVar(). В качестве четвертого аргумента этого метода указывается тип одного из фильтров, перечисленных в табл. Б.1. Ниже приведен характерный пример применения фильтра ARRAY в классе JRequest.

```
$data = JRequest::getVar('jform', array(), 'post', 'array');
```

Следует иметь в виду, что по умолчанию тип фильтра для метода getVar() отсутствует, а следовательно, при вызове этого метода нужно всегда указывать конкретный тип фильтра. В противном случае фильтрация данных не состоится. В состав класса JRequest входят следующие служебные методы, допускающие фильтрацию данных: getInt(), getUInt(), getFloat(), getBool(), getWord(), getCmd() и getString().

Класс JInput считается относительно новым, поскольку он был внедрен в версии 11.1 платформы Joomla, заменив собой класс JRequest. В этом классе имеется метод get(), используемый таким же образом, как и метод JRequest::getVar(). В качестве третьего аргумента этого метода указывается тип фильтра. Ниже приведены три равнозначных примера применения фильтра CMD, название которого выделено полужирным.

```
$x = JRequest::getVar('option', 'post', 'default', 'cmd');
$x = JRequest:: getCmd ('option', 'default');
$x = JFactory::getApplication()->input->get('option', 'default', 'cmd');
```

Начиная с версии 12.1 платформы Joomla класс JRequest считается устаревшим и непригодным к употреблению. А в системе управления информационным наполнением (CMS) веб-сайтов версии Joomla 2.5 этот класс применяется более чем в 700 местах. Но и здесь ему на смену в конечном итоге придет класс JInput.

Непосредственное применение класса `JFilterInput` для фильтрации данных

Классом `JFilterInput` можно пользоваться непосредственно для фильтрации данных в любом приложении Joomla. Например, в приведенной ниже строке кода основной текст сообщения электронной почты просеивается с помощью фильтра `STRING`.

```
$message_body = JFilterInput::getInstance()->clean($message_body,
↳'string');
```

В данном примере сначала вызывается метод `getInstance()` для получения объекта типа `JInput`, а затем метод `clean()` для фильтрации нужного значения.

Приложение В

Методы из класса JHtml

В папке libraries/joomla/html содержится целый ряд полезных методов для разработки приложений Joomla!. Некоторые из них вкратце описываются в этом приложении. Следует, однако, иметь в виду, что все эти методы статические. Дополнительные сведения о них можно найти по адресу http://api.joomla.org/li_Joomla-Platform.html.

Вызов методов из класса JHtml

Методы из класса JHtml предназначены для удобного применения разработчиками и прочими лицами, не обладающими достаточным опытом программирования на языке PHP. Их удобство состоит, в частности, в том, что они могут быть автоматически импортированы по мере надобности. Синтаксис обращения к ним довольно прост, как показано в приведенном ниже примере.

```
<?php echo JHTML::_('string.truncate', $item->introtext,
↳$params->get('introtext_limit')); ?>
```

В данном примере метод _() из класса JHtml, объявляемого в файле libraries/joomla/html/html.php, служит для вызова метода truncate() из класса JHtmlString, объявляемого в файле libraries/joomla/html/html/string.php. В качестве первого аргумента метода _() указывается идентификатор вызываемого метода. Перед его именем может быть указан через точку любой префикс, обозначающий имя класса, находящегося по умолчанию в папке libraries/joomla/html/html. В данном примере в первой части идентификатора указан префикс **string**, обозначающий класс JHtmlString, а во второй части — имя вызываемого метода, truncate(). Если же префикс отсутствует,

то вызывается метод из самого класса JHtml. Например, в приведенной ниже строке кода вызывается метод **link()** из класса JHtml.

```
<?php echo JHtml::_( 'link',$url, $v, array('target' => 'helpFrame'));?>
```

Остальные аргументы, передаваемые методу _(), передаются далее вызываемому методу.

Специальные классы типа JHtml

С помощью метода addIncludePath() нетрудно добавить свои собственные классы типа JHtml. Например, в приведенной ниже строке кода из компонента administrator/com_admin добавляется путь administrator/com_admin/helpers/html к включаемым файлам.

```
JHtml::addIncludePath(JPATH_COMPONENT.'/helpers/html');
```

Это означает, что специальные классы типа JHtml можно добавить в данную папку и вызвать их с помощью метода JHtml::_(). В этом случае они будут загружены автоматически, как и базовые классы типа JHtml. Например, в приведенной ниже строке кода, взятой из файла administrator/components/com_admin/views/sysinfo/tmpl/default_directory.php, вызывается метод message() из специального класса JHtmlDirectory, находящегося в папке administrator/com_admin/helpers/html.

```
<?php echo JHtml::_('directory.message',$dir,$info['message']);?>
```

Методы из класса JHtml

В этом разделе кратко описаны наиболее употребительные методы из класса JHtml.

Метод link()

Создает элемент привязки HTML-документа. У него имеются три следующих аргумента: URL, текст элемента привязки и дополнительный ассоциативный массив атрибутов данного элемента. Ниже приведен характерный пример вызова данного метода.

```
$output = JHtml::_('link', JRoute::_($url), $text, $attribs);
```

Метод image()

Создает элемент img разметки изображения в коде HTML. У него имеются пять следующих аргументов: файл изображения для атрибута src, атрибут alt, ассоциативный массив с другими атрибутами, логическое значение, обозначающее, является ли путь к файлу относительным, а также логическое значение, обозначающее, следует ли искать файл замещения в папке с шаблонами. Ниже приведен характерный пример вызова данного метода.

```
$text = JHtml::_('image','system/new.png', JText::_('JNEW'), NULL, true);
```

Метод stylesheet()

Создает элемент разметки <link rel="stylesheet" style="text/css" />. У него имеются пять следующих аргументов: путь к файлу с вложенными таблицами стилей CSS,

массив дополнительных атрибутов, логическое значение, обозначающее, указан ли путь к файлу относительно папки с мультимедийными файлами, логическое значение, обозначающее, следует ли искать файл замещения в папке с шаблонами, а также логическое значение, обозначающее, следует ли включать характерные для браузера файлы с вложенными таблицами стилей CSS. Ниже приведен характерный пример вызова данного метода.

```
JHtml::_('stylesheet',$component.'/administrator/categories.css', array(),
↳true);
```

Метод `script()`

Создает элемент разметки `script` в коде HTML. У него имеются пять следующих аргументов: путь к файлу сценария JavaScript; логическое значение, обозначающее, следует ли загружать интегрированную среду MooTools, логическое значение, обозначающее, указан ли путь к файлу относительно папки с мультимедийными файлами, логическое значение, обозначающее, следует ли искать файл замещения в папке с шаблонами, а также логическое значение, обозначающее, следует ли включать характерные для браузера файлы сценариев JavaScript. Ниже приведен характерный пример вызова данного метода.

```
JHtml::_('script','media/popup-imagemanager.js', true, true);
```

Метод `calendar()`

Создает всплывающий календарь, используя поведение `calendar` из интегрированной среды MooTools. У него имеются пять следующих аргументов: текущая дата, имя текстового поля, идентификатор текстового поля, формат даты, а также дополнительный массив атрибутов. Ниже приведен характерный пример вызова данного метода.

```
return JHtml::_('calendar', $this->value, $this->name, $this->id, $format,
↳$attributes);
```

Метод `date()`

Отображает дату в заданном формате и часовом поясе. У него имеются четыре следующих аргумента: отображаемая дата, формат, часовой пояс, а также логическое значение, обозначающее, следует ли использовать григорианский календарь. Ниже приведен характерный пример вызова данного метода.

```
<?php echo JHtml::_('date',$item->created, JText::_('DATE_FORMAT_LC4')); ?>
```

Классы из папки `html`

По умолчанию метод `JHtml::_()` обращается для поиска классов и методов к папке `libraries/joomla/html/html`. В этом разделе вкратце описываются некоторые из наиболее употребительных классов, находящихся в данной папке. Имя всех этих классов начинается буквами `JHtml`.

Класс `JHtmlBatch`

Реализует групповые операции на экранах диспетчеров в административной части вебсайта. В него входят методы `access()`, `item()` и `language()` для разметки групповых

операций, изменяющих уровни доступа, категории и язык. Ниже приведены характерные примеры вызова методов из данного класса.

```
<?php echo JHtml::_('batch.access');?>
<?php echo JHtml::_('batch.language'); ?>
```

Класс JHtmlBehavior

Реализует различные виды поведения JavaScript. Подробнее о них см. в главе 12.

Класс JHtmlCategory

Возвращает кешированный список категорий для заданного расширения. Служит для создания раскрывающихся списков, из которых выбираются категории на экранах диспетчеров в административной части веб-сайта. Ниже приведен характерный пример обращения к этому классу.

```
<?php echo JHtml::_('select.options', JHtml::_('category.options',
↳'com_banners'), 'value', 'text', $category);?>
```

Класс JHtmlContent

У этого класса имеется единственный метод `prepare()`, инициирующий событие `onContentPrepare` для обработки в соответствующем подключаемом модуле. Этот класс служит для инициирования данного события по отношению к разному содержимому, например, контактной информации, специальных HTML-модулей и прочего, но кроме статей. Ниже приведен характерный пример обращения к этому классу.

```
$module->content = JHtml::_('content.prepare', $module->content);
```

Класс JHtmlContentLanguage

У этого класса имеется единственный метод `existing()`, составляющий список имеющихся языков. Этот класс служит для создания раскрывающегося списка, из которого выбирается язык. Ниже приведен характерный пример обращения к этому классу.

```
<?php echo JHtml::_('select.options', JHtml::_('contentlanguage.existing',
↳true, true), 'value', 'text', $this->state->get('filter.language'));?>
```

Класс JHtmlEmail

У этого класса имеется единственный метод `cloak()`, заменяющий адрес электронной почты ссылкой, где этот адрес скрывается. Ниже приведен характерный пример обращения к этому классу.

```
$item->email_to = JHtml::_('email.cloak', $item->email_to);
```

Класс JHtmlForm

У этого класса имеется единственный метод `token()`, отображающий скрытое поле маркера формы для снижения риска от попыток взлома типа межсайтовой подделки запросов. Ниже приведен характерный пример обращения к этому классу.

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

Класс JHtmlGrid

У этого класса имеется целый ряд методов, используемых для создания элементов управления на экранах различных диспетчеров в административной части веб-сайта. Ниже перечислены методы этого класса.

- boolean() — создает элементы управления, переключаемые щелчком кнопкой мыши, например, для разрешения или отмены действий пользователя.
- sort()— создает ссылки, выбираемые щелчком кнопкой мыши, в заголовках столбцов, позволяющие сортировать таблицу по выбранному заголовку.
- id() — создает флажки для элементов сетки.
- order() — создает пиктограмму для сохранения переупорядоченной таблицы на экране диспетчера.

Класс JHtmlImage

У этого класса имеются два метода, admin() и site(), загружающие изображение из замещения шаблона или изображение, выбираемое по умолчанию.

Класс JHtmlJGrid

Предоставляет целый ряд методов, используемых для оформления пользовательского интерфейса на экранах диспетчеров в административной части веб-сайта. Ниже перечислены методы этого класса.

- checkedOut() — отображает пиктограмму, щелчком на которой уполномоченный пользователь может снять элемент с регистрации.
- isDefault() — указывает, является ли элемент (например, язык или шаблон) установленным по умолчанию.
- published() — отображает пиктограмму, щелчком на которой уполномоченный пользователь может переключить элемент в состояние "опубликовано".
- publishedOptions() — отображает массив стандартных параметров фильтрации состояния "опубликовано".

Класс JHtmlList

Предоставляет методы для составления различных списков. В частности, метод ordering() служит для составления списка, из которого выбираются способы упорядочения при редактировании элемента (например, контактной информации).

Класс JHtmSelect

Предоставляет методы для составления списков выбора в формате HTML. Ниже перечислены методы этого класса.

- genericList() — создает список выбора в формате HTML.
- groupedList() — создает список выбора в формате HTML для вложенного массива. Используется, например, для отображения альтернативных компоновок, сгруппированных по шаблону.

- `options()` — создает дескрипторы `option` для списка выбора в формате HTML.
- `radioList()` — создает список кнопок-переключателей в формате HTML.

Класс `JHtmSliders`

Предоставляет методы для создания элементов управления слайдерами, используемых в административной части веб-сайта для выбора параметров. Ниже перечислены методы этого класса.

- `start()` — создает панели и загружает код сценария JavaScript для управления слайдерами.
- `panel()` — открывает новую панель.
- `end()` — закрывает текущую панель.

Класс `JHtmString`

Предоставляет следующие методы для работы с символьными строками в компоновках.

- `abridge()` — сокращает текстовую строку, показывая первую и последнюю ее части, разделяемые многоточием (например, строка "Длинное, слишком длинное название" превращается в строку "Длинное .... название").
- `truncate()` — обрезает символьную строку до заданной длины и закрывает любые открытые дескрипторы HTML.

Класс `JHtmTabs`

Предоставляет методы для создания вкладок, например, на экране Global Configuration. Ниже перечислены методы этого класса.

- `start()` — создает панели с вкладками и загружает код сценария JavaScript для управления ими.
- `panel()` — открывает новую панель.
- `end()` — закрывает текущую панель.

Приложение Г

Словарь специальных терминов

Apache (HTTP-сервер Apache). Программа управления веб-сервером, применяемая на большинстве веб-сайтов на Joomla. *См.* также **LAMPP**.

API. Прикладной программный интерфейс, содержащий стандартные средства для разработки прикладных программ. В Joomla под API подразумеваются открытые поля и методы всех определенных классов, особенно на платформе Joomla. Программы сторонних разработчиков опираются на API, и поэтому изменения в данный интерфейс следует вносить как можно реже, предупреждая заранее.

cron. Служебная программа ОС Linux для планирования работы прикладных программ. Может использоваться вместе с платформой Joomla для автоматического выполнения ее приложений с интерфейсом командной строки по определенному графику.

CSS. Вложенные таблицы стилей, описывающие стилевое оформление веб-страниц, в том числе компоновку, шрифты, цвета и графику. Как правило, команды стилевого оформления размещаются в отдельных файлах с расширением `.css`, включаемых вместе с шаблоном.

Eclipse. Интегрированная среда разработки с открытым кодом, включая и версию для работы с проектами на языке PHP. Применяется многими разработчиками Joomla.

FOSS. Свободно доступное программное обеспечение с открытым кодом. Программное обеспечение вроде Joomla, которым можно пользоваться бесплатно. Как правило, такое программное обеспечение распространяется по специальной лицензии (*см.*, например, **GPL**), требующей свободной доступности и от любого производного программного обеспечения.

GPL. Общедоступная лицензия на программное обеспечение, разрабатываемое во многих проектах с открытым исходным кодом (*см.* также **FOSS**). Например, Joomla распространяется по лицензии GPL2.

HTML. Язык гипертекстовой разметки, специально предназначенный для компоновки веб-страниц. Все веб-страницы отображаются в браузере размеченными в текстовом формате HTML, а браузер интерпретирует код HTML и соответственно форматирует страницу.

HTML5. Новый стандарт для разметки веб-страниц, который в конечном итоге должен прийти на смену XHTML. Обеспечивает усовершенствованную обработку мультимедийных (аудио и видео) файлов и поддерживается на большинстве мобильных устройств. Пока еще находится на стадии разработки, но в ближайшем будущем должен стать общепринятым стандартом для разработки большинства веб-сайтов.

IIS. Веб-сервер Microsoft Internet Information Services, работающий под управлением ОС Windows. Может быть использован вместо HTTP-сервера Apache для Joomla.

JavaScript. Язык программирования, предназначенный для создания сценариев, выполняющихся в браузере на стороне клиента. Широко применяется при разработке веб-приложений. В Joomla язык JavaScript служит для усовершенствования пользовательского интерфейса, предоставляя доступ к функциональным средствам AJAX.

LAMPP. Обозначает Linux Apache MySQL Perl/PHP/Python, т.е. набор программ, предназначенных для поддержки нормальной работы многих динамических веб-сайтов. Большинство веб-сайтов на Joomla работают с базой данных SQL и веб-сервером Apache, и все они разработаны на языке PHP. А теперь важной составляющей их разработки стал и язык JavaScript.

MySQL. Тип базы данных, применяемой в большинстве установок Joomla и на динамических веб-сайтах по всему миру.

NetBeans. Интегрированная среда разработки с открытым кодом, поддерживающая программирование на языке PHP.

PHP. Язык программирования, на котором написана бо́льшая часть исходного кода Joomla. Этот язык программирования чаще всего употребляется и при разработке веб-сайтов.

SQL. Язык структурированных запросов, применяемый для составления запросов к базам данных SQL. Включает в себя язык определения данных (ЯОД — DDL) и язык манипулирования данными (ЯМД — DDM).

XHTML. Расширяемый язык гипертекстовой разметки. Представляет собой более структурированную разновидность HTML на основе XML. Большинство страниц веб-сайтов на Joomla размечены в формате XHTML.

XML. Расширяемый язык разметки. XML-файлы используются для установки Joomla, а также компоновки параметров и форм типа `JForm`.

Альтернативный пункт меню. Видоизмененный базовый пункт меню, в котором используется XML-файл замещения и файл компоновки для изменения режима отображения и функционирования этого пункта меню.

Ассоциативный массив. Массив, применяемый в языке PHP для хранения пар "ключ–значение" (например, `name => 'George', title => 'President'`).

Булев, или логический, тип данных. Тип переменной в языке PHP, которая может принимать одно из двух значений: `true` (истина) или `false` (ложь).

Веб-сервер. Программа, обеспечивающая связь между браузером на машине клиента и сервером, где хранятся файлы веб-сайта и его базы данных. В Joomla предусмотрена работа на двух типах веб-серверов: Apache и Microsoft IIS.

Волшебные константы. Специальные константы, имеющие в языке PHP особое назначение. Например, в константе `__FILE__` всегда устанавливается полный путь к файлу PHP, выполняющемуся в настоящий момент.

Волшебные методы. Специальные методы, вызываемые в языке PHP автоматически. К их числу относится метод `__construct()`, вызываемый при создании нового объекта, а также метод `__call()`, вызываемый всякий раз, когда происходит обращение к несуществующему методу.

Доработка базового кода. Изменение файла, содержащего базовый код Joomla. Допускается, но не рекомендуется, поскольку внесенные изменения могут быть утрачены в процессе обновления кода. Как правило, для выполнения требующейся задачи в Joomla могут вноситься изменения без доработки базового кода.

Заготовка. В Joomla обозначает значение, состоящее из идентификатора элемента и его псевдонима, разделенных двоеточием (например, `123:my-article-alias` или `234:my-category-alias`). Заготовки служат в качестве вспомогательного средства при формировании удобных для поиска URL.

Замещение компоновки. Файл, позволяющий изменить порядок отображения модулей или компонентов. Файлы замещения компоновки размещаются в отдельных папках и могут использоваться вместо стандартных файлов компоновки. В отличие от замещений шаблонов, которые замещают файлы компоновки с тем же именем, замещения компоновки являются дополнительными файлами компоновки.

Замещение шаблона. Представляет собой файл компоновки, размещаемый в отдельной подпапке шаблона. Если такой файл обнаруживается, то он замещает файл стандартной компоновки по данному шаблону.

Замещение. *См.* **Замещение шаблона**, **Замещение компоновки**, **Языковое замещение**, а также **Альтернативный пункт меню**.

Индексированный массив. Массив, применяемый в языке PHP для хранения данных по порядку номеров, начиная с нуля.

Интегрированная среда разработки (IDE). Набор программ, повышающих производительность труда разработчиков программного обеспечения. В частности, IDE способны автоматически завершать ввод операторов и команд языка программирования, обнаруживать синтаксические ошибки, отлаживать программы и переходить к родительскому классу или его методу. К числу наиболее употребительных IDE для разработки программ на языке PHP относятся Eclipse и NetBeans.

Класс. Программный код, объявляющий класс в объектно-ориентированном программировании. Класс определяет тип объекта и может содержать поля и методы. В Joomla каждый класс, как правило, определяется в отдельном файле исходного кода на языке PHP.

Компонент. Разновидность расширения Joomla. Как правило, компоненты составляют основную часть страницы веб-сайта. К числу базовых компонентов Joomla относятся `com_content` (статьи в пользовательской части веб-сайта) и `com_users` (диспетчер пользователей (компонент User Manager) в административной части веб-сайта). На одной странице веб-сайта, построенного на Joomla, обычно располагается лишь один компонент.

Конструктор. Специальный (называемый "волшебным" в языке PHP) метод, вызываемый для создания нового объекта. Обычно вызывается с помощью оператора `new`, как, например: `$reg = new JRestistry()`.

Контроллер. Часть шаблона проектирования MVC (Модель–Представление–Контроллер), применяемого в базовых компонентах Joomla. Контроллер выполняет задачи по командам пользователя.

Маршрутизатор. Программа, преобразующая в Joomla массив команд в URL, и наоборот. Как правило, маршрутизатор содержит два метода: `build()` и `parse()`. Метод `build()` принимает массив команд и возвращает объект URL. А метод `build()`, наоборот, превращает объект URL в массив команд.

Метод. Другое название функции, определяемой в классе. Зачастую используется для обращения к функциям внутри класса.

Модель. Одна из трех составляющих шаблона проектирования MVC (Модель–Представление–Контроллер). Как правило, модель содержит логику конкретной предметной области (так называемую бизнес-логику) и взаимодействует с базой данных.

Модель–Представление–Контроллер (MVC). Шаблон проектирования, применяемый в разработке базовых компонентов Joomla. Является стандартным для ООП шаблоном проектирования, в котором различные логические функции разделяются на разные классы для повышения удобства сопровождения программного обеспечения.

Модификатор доступа. Одно из ключевых слов — `public`, `protected` или `private`, — используемых в языке PHP для управления доступом к полям и методам класса. Открытые поля и методы с модификатором доступа `public` доступны в любом месте программы. Защищенные поля и методы с модификатором доступа `protected` доступны только в текущем классе, а также в родительском или производном от него классе. А закрытые поля и методы с модификатором доступа `private` доступны только в текущем классе. Следует, однако, иметь в виду, что закрытые поля и методы не наследуются производными классами.

Модуль. Разновидность расширения Joomla. Модули, как правило, мелкие, простые и легковесные. Одна страница веб-сайта на Joomla обычно содержит несколько модулей.

Набор данных. Данные, возвращаемые по команде `SELECT` из запроса SQL. Содержат строки и столбцы, извлекаемые из таблицы базы данных SQL.

Не рекомендованные к применению методы или поля. Устаревшие методы или поля, которые предполагается исключить из прикладного интерфейса API. Рекомендация не применять такие методы или поля предупреждает пользователей API о том, что они должны изменить свой код, чтобы не пользоваться этими частями API в последующих выпусках.

Объектно-ориентированное программирование (ООП). Методика программирования, основанная на классах, объектах и методах. Разработка Joomla опирается на ООП, что дает главное преимущество для работы с прикладными программами Joomla.

Оператор тождественности. В языке PHP этот оператор обозначается тремя знаками равенства (`===`) и проверяет сравниваемые значения не только на равенство, но и на принадлежность к одному и тому же типу данных (например, строковому, целочисленному и т.д.).

Параметры. Иначе называются вариантами выбора и являются установками, которые администратор может задавать и настраивать, чтобы изменить поведение веб-сайта на Joomla. Базовые расширения Joomla допускают установку многих параметров, позволяющих управлять различными свойствами веб-сайта, не прибегая непосредственно к программированию.

Переопределение методов. Способ создания в подклассе метода с таким же самым именем, как и в родительском классе. Если такой метод переопределен, то он выполняется вместо метода из родительского класса. Переопределяемый метод должен иметь такую же сигнатуру, как и метод из родительского класса. Тем не менее метод из родительского класса можно вызывать из переопределяемого метода с помощью ключевого слова `parent`, как, например, `parent::display()`.

Переопределение. *См.* **Переопределение методов.**

Подключаемый модуль. Разновидность расширения Joomla. Подключаемые модули выполняются на предопределенных стадиях в цикле выполнения команд Joomla, когда наступают соответствующие события, с целью изменить обычное поведение.

Представление базы данных. Запрос, который сохраняется в базе данных SQL и может служить в качестве таблицы. Что касается версии Joomla 2.5, то в основной базе данных представления не используются.

Представление по шаблону MVC. Одна из составляющих шаблона проектирования MVC (Модель–Представление–Контроллер), применяемого в базовых компонентах Joomla. Представление отвечает за отображение компонента в окне браузера.

Псевдоним. Поле, используемое в Joomla! для таких элементов, как статьи, пункты меню и контактная информация, при формировании удобных для поиска URL. Поля псевдонимов могут быть введены вручную или созданы автоматически, исходя из названия вводимого элемента, из которого удаляются все символы, кроме строчных букв, цифр и тире. Например, из названия статьи `"The Joomla Project"` образуется псевдоним `"the-joomla-project"`.

Расширение. В Joomla под расширением подразумевается набор программ, обеспечивающих определенный ряд функций веб-сайта. К разновидностям расширений относятся модули, компоненты, подключаемые модули, языковые расширения и шаблоны. Расширениями могут быть также сторонние программы, расширяющие функциональные возможности базового пакета Joomla (например, программы, доступные на веб-сайте, под названием Joomla Extensions Directory (JED — Каталог расширений Joomla; `http://extensions.joomla.org`).

Реорганизация кода. Исправление одной или нескольких программ с целью улучшить их внутреннюю структуру, не изменяя их внешнее поведение (прикладной интерфейс API). Например, можно переделать тело метода, чтобы повысить его эффективность, сохранив в то же время его сигнатуру и функции.

Связывание методов в цепочку. Использование результатов выполнения одного метода для непосредственного вызова другого метода. Внедрено в версии 5 языка PHP.

Событие, обрабатываемое подключаемым модулем. Предопределенная стадия в цикле выполнения команд Joomla, на которой запускается на выполнение подключаемый модуль.

Столбец таблицы в базе данных. Содержит один тип данных в таблице базы данных. Например, в таблице `#__content` имеется заголовок `title` с названием каждой статьи, хранящейся в базе данных.

Строка таблицы в базе данных. Обычно содержит сведения об одном элементе, например, статье или пользователе.

Тернарный оператор. Синтаксическая конструкция, применяемая в языке PHP, для присваивания переменной значения в зависимости от истинности или ложности задаваемого условия, как в приведенном ниже примере.

```
$a = ($b == $c) ? $d : $e;
```

Файл с расширением `.ini`. Служит для языковой поддержки в Joomla.

Функция. Программный код на языке PHP, выполняющий определенные действия. Иногда еще называется методом, если определена в классе. Большая часть исходного кода Joomla находится в функциях или методах.

Число посещений. Количество обращений к конкретной странице или ее элементу (например, статье или рекламе с крупным заголовком). В Joomla отслеживается число посещений многих разновидностей элементов веб-сайта.

Шаблон. Разновидность расширения Joomla, управляющая всеми свойствами, определяющими внешний вид веб-сайта. В частности, шаблон управляет компоновкой страниц, а также шрифтами, цветами и графикой.

Экранирование текста. Метод или функция, заменяющие или удаляющие специальные символы для безопасного отображения символьной строки на веб-странице. Например, метод `htmlspecialchars()` заменяет знак **<** на ****, чтобы он отображался на веб-странице как знак **<**, но не имел специального значения открывающего дескриптора.

Язык манипулирования данными (ЯМД — DDM). Состоит из команд запросов базы данных SQL, обрабатывающих данные в таблицах базы данных (например, `SELECT`, `UPDATE` или `DELETE`).

Язык определения данных (ЯОД — DDL). Состоит из команд запросов базы данных SQL, создающих и изменяющих структуру таблиц базы данных (например, `CREATE TABLE` или `ALTER TABLE`).

Языковое замещение. Дополнительный файл, позволяющий администратору замещать текстовые строки на любом языке мира в содержимом веб-сайта на Joomla. Как правило, имеются два файла языкового замещения: один — для пользовательской части веб-сайта, другой — для административной.

Языковый ключ. Название отдельного фрагмента текста на конкретном языке в приложении Joomla. Во время выполнения программ, написанных для Joomla, языковые ключи служат для подстановки текстовых строк на том языке мира, который выбран в настоящий момент.

Языковый пакет. Устанавливаемое языковое расширение, позволяющее просматривать содержимое веб-сайта Joomla на разных языках мира. Языковые пакеты могут быть предназначены как пользовательской, так и для административной части веб-сайта.

Предметный указатель

J

JavaScript
  запросы AJAX, составление, 488
  ненавязчивый, стандарты, 490
  особенности, 488
  реализация в Joomla, 490
  стандартные виды поведения в Joomla, описание, 491; 512
  сценарии
    включение в состав HTML-страниц, 488
    выполнение, 488
    обрабатываемые события, 488

P

PHP
  альтернативный синтаксис, 132; 553
  волшебные методы, 567
  команды
    clone, 104
    defined, 179
    die, 565
    echo, 116
    include_once, 99; 100
    jimport, 100
    preg_replace, 108
    require, 242
    require_once, 100; 219; 242
    unset, 296
  комментарии, обозначение, 546
  константы
    определение, 556
    специальные или волшебные, 557
  конструктор, назначение, 281
  массивы
    индексированные и ассоциативные, 555
    объявление, 555
    применение, 556
  обработка исключений в блоках try/catch, 320
  операторы
    арифметические, 548
    выбора switch, 551
    логические, 549
    строковые, 548
    цикла, разновидности, 551
  переменные
    задание или приведение типов, 548
    инициализация, 244
    область действия, 192; 554
    обозначение и объявление, 554
  присваивание и передача объектов по ссылке, 182
  символьные строки
    обработка, 556
    определение, 556
  сочетание элементов разметки PHP и HTML, 130
  специальные символы, 546
  структура файлов, 545
  тернарный оператор, 168; 551
  условные операторы
    разновидности, 549
    нелогические значения, применение, 186
  функции
    array_keys(), 444
    array_merge(), 347
    array_reverse(), 342
    clone(), 406
    close(), 444
    dirname(), 524; 557
    explode(), 187
    file_exists(), 557
    fopen(), 443
    fputcsv(), 444
    get_defined_vars(), 565
    header(), 443
    htmlspecialchars(), 229; 407
    implode(), 245
    isset(), 168; 219
    md5(), 323
    ob_start(), 109; 309
    parse_ini_file(), 171
    назначение, 557
    serialize(), 323
    strip_tags(), 148
    substr(), 147
    var_dump(), 146
    область действия переменных, 558
    структура, 557

передача значений по ссылке, 558
сигнатура, 558
явное указание типов, 533

Б

Базовый код
доработка, 34
определение, 34
Базы данных
MySQL
выражения и функции, 472
обращение с помощью phpMyAdmin, 457
одинарные кавычки, применение, 456
SQL
логические единицы хранения данных, 448
назначение, 448
сценарии, 449
в Joomla
краткое описание таблиц, 120
назначение, 120
соединение
левое и внутреннее, 249; 464
назначение, 249

В

Веб-приложения
выполнение из командной строки в Joomla CMS, 543
разновидности, 522
разработка на плотформе Joomla, примеры, 523–528
Веб-программирование
контроль и проверка команд, 94
отличительные особенности, 91
сохранение состояния программы, 92
Версии Joomla
2.5, нововведения, 26
изменения и отличия, 26
Включение файлов PHP, особенности, 100; 230
Внутренний механизм Joomla
библиотеки, 80
защита файлов на веб-сервере, 78
кеширование данных, 75
пользовательская и административная части, 74
процесс установки, 79
типы пунктов меню и компоненты, 76
файлы языковой поддержки, 86
цикл выполнения команд, 96

Г

Глобальные объекты, 119

Д

Древовидные структуры
назначение, 511
отображение, 512
построение, 511

З

Замещения
компоновки
модулей, пример создания, 142
компонентов, пример создания, 156
назначение, 36
стилевой рамки модуля, 164
языковые
назначение, 79
размещение, 169
применение, 172
Запросы SQL
выражения, 472
в Joomla, составление, 475
на ввод
обработка в классе JDatabaseQuery, 482
формирование в классе JDatabaseQuery, 480
по команде INSERT, составление, 471
на выборку
возврат результатов методами класса JDatabase, 484
формат команды SELECT, 231
обработка в классе JDatabaseQuery, 482
формирование в классе JDatabaseQuery, 476
сортировка результатов, 232
по команде SELECT, составление, 462
на обновление
обработка в классе JDatabaseQuery, 482
формирование в классе JDatabaseQuery, 481
по команде UPDATE, составление, 470
на объединение
отсутствие поддержки в классе JDatabaseQuery, 482
по команде UNION, составление, 472
на удаление
обработка в классе JDatabaseQuery, 482
формирование в классе JDatabaseQuery, 481
по команде DELETE, составление, 471
обработка в Joomla, 482
составные
назначение и формирование, 468
формирование в классе JDatabaseQuery, 479

И

Интегрированные среды JavaScript

MooTools
расширения, применение, 513
внедрение в Joomla, 490
преимущества, 489
другие, применение в Joomla, 518

К

Классы
ContentHelperRoute, методы, 253
JAdministrator, применение, 102
JApplication
расширение, 178
методы, 103; 178
JComponentHelper, методы, 107
JController
рефлексия, 281
методы, 277
импорт, 275
конструктор, 282
сопоставление задач с методами, 280
JControllerAdmin
методы, 282; 284; 288
конструктор, 281
JControllerForm, конструктор и методы, 289
JDatabase
назначение, 475
методы, 233; 246; 247; 478; 484
JDatabaseMySQLi, методы, 121
JDatabaseQuery
назначение, 475
методы, 232; 233; 250; 476
обработка запросов SQL, 482
объявление абстрактным, 369
формирование запросов SQL, 479; 481
применение, 232; 476
JDocumentFeed, назначение, 349
JDocumentHTML, методы, 114
JFactory
назначение, 119
методы, 245
JFeedItem, назначение, 349
JFilterInput, фильтрация данных, 259; 572
JForm
методы, 218
фильтрация данных, 258; 570
средства проверки достоверности, 256
применение, 210; 318; 385
сохранение объектов в памяти, 323
сокрытие полей, 241
JFormField, специального типа
применение, 492
создание, 404
JFormRule, расширения, 260
JHtml
вызов методов, 154; 574
методы поведения, 490
фильтрация данных, средства, 312
описание методов, 574
специального типа, вызов и описание, 574–575
JInput
методы, 571
фильтрация данных, 571
JLoader
методы, 101; 102; 219; 242
применение, 101
JModelAdmin
методы, 293
описание, 189
JModelForm, методы, 299; 319
JModuleHelper, методы, 228
JoomproSubsCategories, описание, 414
JoomproSubsController, описание, 356; 401
JoomproSubsControllerSubManager, описание, 358
JoomprosubsControllerSubscription, описание, 377
JoomproSubsControllerSubscription, описание, 420
JoomprosubsHelper, описание, 364
JoomproSubsModelCategory, описание, 410
JoomproSubsModelCSVReport, описание, 439
JoomproSubsModelForm, описание, 430
JoomproSubsModelSubManager, описание, 365
JoomproSubsModelSubscription, описание, 381
JoomprosubsTableSubscription, описание, 391
JoomproSubsViewCategory, описание, 405
JoomproSubsViewForm описание, 422
JoomproSubsViewSubManager, описание, 361
JoomproSubsViewSubscription, описание, 379
JPlugin, назначение, 179
JPluginHelper, применение, 218
JRequest
методы, 191; 204
фильтрация данных, 571
JResponse, методы, 116
JRouterSite, назначение и методы, 345
JSite
назначение, 178

методы, 103; 177
применение, 102
JString, методы, 150; 556
JTable
назначение, 121
методы, 300
JTableNested, замещение, 220
JToolBarHelper, методы, 438
JView
расширение, 332
методы, 308; 332
JWeb, назначение и методы, 528
JCli, назначение и методы, 526
PlgContentJoomla, методы, 189
plgSystemSef
назначение, 177
объявление, 179
SubscriptionMonitor, назначение и методы, 532; 533; 541
WeblinksController
назначение и методы, 276; 288; 305
реализация контроллера, 276
WeblinksControllerWeblink
назначение и методы, 289
WeblinksControllerWeblinks
назначение и методы, 288
WeblinksModelWeblink, назначение и методы, 292; 299; 334
WeblinksModelWeblinks, назначение и методы, 292
WeblinksTableWeblink, назначение и методы, 300
WeblinksViewWeblink, назначение и методы, 317
абстрактные, применение, 243
базовые Joomla, загрузка, 219
библиотеки Joomla, основные задачи, 46
конструкторы, назначение, 561
контроллеров, обозначение, 292
маршрутизаторов, назначение и методы, 103; 337
модификаторы доступа к полям и методам, 284
модулей, вспомогательные, 230
назначение, 561
обозначение, 179
подключаемых модулей, обозначение, 175
расширение и наследование, 562
стандартные, 562
статические методы и поля, 564

Команды SQL
DDL типа, описание, 450–457
DML типа, описание, 462–472
общие типы, 449
Компоненты
Weblinks, административная часть
вспомогательные классы и методы, 273
меню компонентов, 272
задачи, классы и методы контроллеров, 288
модели, основные функции, 292
контроллеры, основные функции, 276
описание, 270
установочные файлы, 271
представления, 305
программные файлы, 270
Weblinks, пользовательская часть
маршрутизация, 337
типы пунктов меню, 334
контроллеры, основные функции, 333
структура папок и файлов, 331
представления и модели, 332
административная часть, пример разработки
таблицы базы данных, 370; 391
файлы языковой поддержки, 394
диспетчер подписок, описание и компоновка экрана, 355–372
модели, 365; 381
контроллеры, 356; 358; 377
структура папок и файлов, 353
представления, 360; 379
составление отчета о подписчиках, 438–444
экранная форма ввода и правки подписки, создание и компоновка, 376–386
административной части веб-сайта, 84; 268
базовые, 41; 76
возможности, 333
назначение, 40; 267
определение, 77
основные функции, 268
параметры, определение, 273
пользовательская часть, пример разработки
файл языковой поддержки, 435
вспомогательный файл, 414
задача подписки, 426
точка входа, 401
форма редактирования, разметка и компоновка, 423–425
модели, 410; 422; 429
контроллеры, 401; 420
компоновка категорий, файлы, 415

структура папок и файлов, 399
представления, 419
пункт меню для выбора представления, создание, 403
экран с благодарностями за подписку, компоновка, 433
пользовательской части веб-сайта, 267
пример разработки
функции и назначение, 351
нереализованные функции, 445
текущий контроль подписок, 528
упаковка, 437
установка и настройка, 396–400
проектное решение, детальная разработка, 352
регистрации пользователя
ввод альтернативного пункта меню, 161
замещение компоновки, 156
изменение внешнего вида, 155
свойства, 268
состав, 40
точка входа, 274
Компоновки
альтернативные
особенности, 159
преимущества, 160
альтернативных пунктов меню, 160
компонентов
замещение, 155
исходные, 156
модулей
замещение, 144
исходные, 144
Контроллеры
взаимодействие с моделями, 292
в компонентах, назначение, 292
по шаблону MVC, основные функции, 269

M

Маршрутизаторы
определение, 103
оптимальное соответствие URL, 339
стандартные в Joomla, 339
Методы
_countItemsInCategory(), назначение, 191
_findItem(), описание, 341
authenticate(), описание, 180; 188
canEditState(), описание, 296
fetchHead(), назначение, 194
getArticleRoute(), описание, 253
getForm(), описание, 319
getHeadData(), назначение, 195
getUsers(), описание, 230
merge(), описание, 406
onAfterRender(), особенности, 179
onUserAuthenticate(), описание, 181
plural(), описание, 194
register()
назначение, 152
применение, 242
renderModule(), описание, 228
truncate()
назначение, 150
применение, 153
запуска подключаемых модулей, реализация, 178
из библиотеки Joomla, 150
контроллеров и моделей, взаимодействие, 300
модификаторы доступа, 284; 563
обозначение, 179
отображения представлений, 306
переопределение, 563
поведения, назначение и применение, 490
получения результатов запросов, 247
расположение аргументов, 192
связывание в цепочку, 307; 479; 566
с синтаксисом подчеркивания, назначения, 154
статические, вызов, 341; 563
Модели
взаимодействие с контроллерами, 287; 292
поддержка выполняемых задач, 293
получение, 286
по шаблону MVC, основные функции, 269
Модули
административной части веб-сайта, 87
в сравнении с компонентами, 225
назначение, 39; 225
особенности, 226
отображения статей автора
функциональные возможности, 265
вспомогательный файл PHP, 243
XML-файл, назначение и создание, 237
языковая поддержка, файлы, 255
исходная компоновка, файл, 254
исходный файл PHP, 241
структура файлов, 236
упаковка, 265
пример создания, 236

проверка достоверности параметров, 256
экранная справка, файл, 264
пользовательской части веб-сайта, 80
последних новостей
настройка, 142
замещение компоновки, 144
последних пользователей, описание, 226–234
применение, 40
рекомендации по разработке, 266
типы, 80

О

Объекты, способы создания, 561
Отладка программ, простая, 565

П

Платформа Joomla
библиотечные пакеты, 90
назначение, 47; 523
подготовка разрабатываемого проекта, 523
причины
выделение в отдельный проект, 522
создания, 89
состав папок, 89
Подготовка к разработке в Joomla
рабочие инструменты
Eclipse, 55
IDE и текстовые редакторы, 54
NetBeans, 61
PHPEclipse, 61
автоматизация тестирования, программы, 70
выбор между Eclipse и NetBeans, 67
текстовые редакторы, 68
коммерчески доступные IDE, 68
контроль версий, программы, 69
построение проектов, программы, 70
установка программных компонентов
в Linux, 52
в Mac OS X, 52
в Windows, 51
общие требования, 49
Подключаемые модули
аутентификации, 180
базовые, 177
замещение классов, 218
импорт, 218
назначение, 37; 173
обработки события onBeforeCompileHead
разработка, 196
назначение, 196
запуск на выполнение, 194
определение, 173
передовой опыт разработки, 222
применение, 39
принцип действия, 174
регистрации
разработка, 201
файлы языковой поддержки, 205
добавление параметров, 215
проверка, 206
усовершенствованные, разработка, 209
связь с событиями в Joomla, 175
системные, SEF, 177
содержимого, 189
типы, 82; 175
удаление средствами phpMyAdmin, 206
упаковка, 208
условные обозначения, 174
Представления
ввода и редактирования подписки, 379
исходная компоновка, 309
категории подписок, 403
ленты новостей, 348
одиночных веб-ссылок, 317
подписок, 419
по шаблону MVC, основные функции, 269
списка веб-ссылок, 306
Приемы программирования, более совершенные
волшебные методы, применение, 567
логические операторы || вместо условных, применение, 566
переменные вместо имен классов и методов, применение, 568
регулярные выражения, применение, 568
связывание методов в цепочку, 566
Проверка достоверности
правила, 259
сообщение об ошибке, 261
средства, 256
формы, 326; 385; 387
Программа PhpMyAdmin
назначение, 457
резервное и обычное копирование базы данных, 461
тестирование и отладка запросов SQL, 457
создание сценариев DDL, 459
Программирование для Joomla
особенности, 33
требования, 34

Пункты меню
альтернативные типы, 160
основные типы, 40

Р

Разделение
запросов на циклы, 279
на пространства имен, 165
Расширения
базовые и сторонние, 36
выбор подходящего типа, 46
определение, 36
присваивание имен, 236
языковые, 42
Режимное окно
выбор статьи, экран, 501
компоновка, 504
конфигурация пользователя, экран, 500
открытие в текущем документе, 499
Реорганизация кода
в Joomla, 303
определение, 303

С

Словарь специальных терминов, 579
События в Joomla
инициирование и обработка, 39
определение, 173
особенность, 39
разновидности, 175
системные, 177
Состав папок Joomla, 74–88
Среда Joomla
база данных MySQL, 32
веб-сервер, назначение, 32
сервер, назначение, 32
способы расширения, 34
шаблоны Joomla, 33
языки
JavaScript, 33
PHP, 32
вложенные таблицы стилей CSS, 33
XHTML и HTML, 32
Средство обнаружения Discover, 140
Стилевая рамка модуля
назначение, 126
определение, 132
оформление, 138
Сцепление, определение, 147

Т

Таблицы базы данных
атрибуты столбцов, описание, 451–456
внешние ключи, 451
главные и вторичные, 464
индексы, применение, 452
краткое описание, 121
набор символов, 453
первичные ключи, 451
предыстории, 475
префиксы имен, 121; 232
разновидности, 472
сокращенное имя, 232
соответствий, 474
справочные, 473
структура
жестко заданная, 448
составление, 472
типы данных, описание, 453
Тестирование программ в Joomla
автоматизация, 70
блочное и системное, 70
Технология AJAX, применение в Joomla, 514

У

Удобные для поиска URL
назначение, 338
построение, 340
применение, 338
синтаксический анализ, 347
Умышленное внесение запросов SQL, предотвращение, 185
Условные обозначения имен, принятые в Joomla, 119

Ф

Файлы в Joomla
включение в программу, команды, 560
объявления
функций, 559
классов, 559
простых сценариев, 560
разновидности, 559
смешанные, 560
Фильтрация данных
в коде HTML, 570
в приложениях Joomla, 570
в Joomla, 258
назначение, 299
стандартных типов, 258
типы фильтров, 569

Форматы
CSV, назначение, 438
JSON, применение, 137
Формы
ввода и редактирования подписок, 386
видоизменение в динамическом режиме, 324
воспроизведение, 325
запрет предъявления, 497
проверка достоверности, 326
сохранение в памяти, 323
Функции
jimport(), описание, 219
WeblinksBuildRoute(), описание, 346
преобразования в формат JSON и обратно, 137
стилевого оформления модулей, 137

Ц

Цикл выполнения команд
воспроизведение страницы, 111
вывод страницы, 116
выполнение компонента, 105
загрузка интегрированной среды Joomla, 101
загрузка файла index.php, 97
маршрутизация URL, 103
начало или продолжение сеанса, 102
определение места расположения файлов, 98

Ш

Шаблоны
Joomla
как расширения, 126
административной части веб-сайта, 88
назначение, 44
базовые, 139
места расположения элементов содержимого, 132
используемые по умолчанию, 44
копирование, 139
описание файла index.php, 127
параметры настройки, 134
папки и файлы, 126
стили оформления модулей, 137
пользовательской части, 83
проектирования
MVC, описание, 269
одиночки и фабрики, 120

Э

Экранирование символов
в HTML, 315
в SQL, 478
в URL, 427
назначение, 478

Я

Языковая поддержка
правила для языковых ключей, 171
режим языка отладки, 169
средство Language Switcher, 252
файлы
разновидности, 86
расположение в папках, 236



Joomla! //Программирование//

Овладев искусством программирования для Joomla!, вы сможете специально настраивать веб-сайты так, как того не позволяет сделать обычная настройка параметров, или устанавливать сторонние расширения. Это авторитетное и подробное руководство по программированию адресовано широкому кругу пользователей Joomla!: от начинающих, не имеющих никакого опыта работы с Joomla!, до опытных программистов, которым требуется быстро освоить последние версии Joomla!: 1.6, 1.7 и 2.5.

В книге представлены самые передовые приемы программирования, опробованные на практике ее авторами, Марком Декстером и Луисом Лэндри, написавшим немалую долю кода новой интегрированной среды Joomla!, подробно описаны многочисленные примеры исходного кода, а также приводится немало полезной справочной информации по всем затрагиваемым темам.

Это справочное и в то же время учебное пособие содержит подобные разъяснения всех вопросов написания программ для Joomla!: от самых простых задач создания замещений шаблонов до самых сложных, связанных с разработкой компонентов по шаблону MVC и приложений на платформе Joomla!.

Узнайте, как:

- Подойти к Joomla! с точки зрения программирования
- Расширить Joomla! модулями, замещениями компоновок и подключаемыми модулями
- Выгодно использовать преимущества новой реализации шаблона проектирования MVC в Joomla!
- Встраивать компоненты в пользовательскую и административную части веб-сайта
- Защищать веб-сайты с помощью новой, эффективной и удобной в работе системы списков контроля доступа
- Работать с базами данных и составлять сложные запросы к ним, в том числе и средствами нового класса JDatabaseQuery
- Расширять возможности веб-сайтов на Joomla! средствами JavaScript и интегрированной среды MooTools
- Разрабатывать специальные структуры категорий для собственных веб-сайтов и расширений Joomla!
- Организовать оперативное обновление своих специальных расширений одним щелчком кнопкой мыши
- Пользоваться платформой Joomla! в качестве среды для разработки приложений

Марк Декстер занимается программированием с 1970-х годов. Он основал коммерческую компанию по разработке программного обеспечения и руководил ею в течение 28 лет, прежде чем выйти в отставку и посвятить себя добровольной работе над открытым программным обеспечением. Он немало потрудился на самых разных участках проекта Joomla!, включая форумы пользователей и разработчиков, инициативную программу Google Summer of Code компании Google, составление документации и экранов оперативной справки. Марк принимал активное участие в процессе разработки Joomla!, начиная с 2009 года, занимаясь исправлением программных ошибок, внедрением дополнительных средств и согласованием выпусков новых версий.

Луис Лэндри написал большую часть интегрированной среды Joomla! для версий 1.5 и 1.6 и в течение пяти лет был одним из главных разработчиков архитетуры Joomla!. Свой первый опыт общения с компьютером он начал с программирования на языке Basic, пользуясь ПК на процессоре Intel 286 в офисе своего отца. С тех пор Луис приобрел опыт программирования на многих языках, включая низкоуровневые типа ассемблера x86, высокоуровневые вроде Java и интерпретируемые, подобные PHP. Он также основал робототехническую группу в своем университете и стал ее активным участником.

ISBN 978-5-8459-1798-0

12035

9 785845 917980

Категория: Программирование
Предмет рассмотрения: Joomla!
Уровень: Начинающий/промежуточный

ВИЛЬЯМС

Издательский дом "Вильямс"
http://www.williamspublishing.com

Addison-Wesley
Pearson Education